# NEC Network Queuing System V (NQSV) User's Guide

# [Management]

## Proprietary Notice

# Preface

This guide explains how to manage NEC Network Queuing System V (NQSV) job management system.

The manual of NEC Network Queuing System V (NQSV) is composed by following user's guides.

| Name | Contents |
|---|---|
| NEC Network Queuing System V (NQSV) User's Guide [Introduction] | This guide explains the overview of NQSV and configuration of basic system. |
| NEC Network Queuing System V (NQSV) User's Guide [Management] | This guide explains the various management functions of the system. |
| NEC Network Queuing System V (NQSV) User's Guide [Operation] | This guide explains the various functions that used by general user. |
| NEC Network Queuing System V (NQSV) User's Guide [Reference] | The command reference guide. |
| NEC Network Queuing System V (NQSV) User's Guide [API] | This guide explains the C programming interface (API) to control NQSV. |
| NEC Network Queuing System V (NQSV) User's Guide [JobManipulator] | This guide explains about the scheduler component : JobManipulator. |
| NEC Network Queuing System V (NQSV) User's Guide [Accounting & Budget Control] | This guide explains the functions of accounting. |


| | |
|---|---|
| February 2018 | 1st edition |
| September 2022 | 15th edition |
| January 2023 | 16th edition |
| March 2023 | 17th edition |
| June 2023 | 18th edition |
| March 2024 | 19th edition |

## Remarks

(1) This manual conforms to Release 1.00 and subsequent releases of the NQSV.

(2) All the functions described in this manual are program products. The typical functions of them conform to the following product names and product series numbers:

| Product Name | product series numbers |
|---|---|
| NEC Network Queuing System V (NQSV) /ResourceManager | UWAF00<br>UWHAF00 (Support Pack) |
| NEC Network Queuing System V (NQSV) /JobServer | UWAG00<br>UWHAG00 (Support Pack) |
| NEC Network Queuing System V (NQSV) /JobManipulator | UWAH00<br>UWHAH00 (Support Pack) |

(3) UNIX is a registered trademark of The Open Group.

(4) Intel is a trademark of Intel Corporation in the U.S. and/or other countries.

(5) OpenStack is a trademark of OpenStack Foundation in the U.S. and/or other countries.

(6) Red Hat OpenStack Platform is a trademark of Red Hat, Inc. in the U.S. and/or other countries.

(7) Linux is a trademark of Linus Torvalds in the U.S. and/or other countries.

(8) Docker is a trademark of Docker, Inc. in the U.S. and/or other countries.

(9) InfiniBand is a trademark or service mark of InfiniBand Trade Association.

(10) Zabbix is a trademark of Zabbix LLC that is based in Republic of Latvia.

(11) All other product, brand, or trade names used in this publication are the trademarks or registered trademarks of their respective trademark owners.

# About This Manual

## Notation Conventions

The following notation rules are used in this manual:

| | | |
|---|---|---|
| Omission Symbol | ... | This symbol indicates that the item mentioned previously can be repeated. The user may input similar items in any desired number. |
| Vertical Bar | \| | This symbol divides an option and mandatory selection item. |
| Brackets | { } | A pair of brackets indicates a series of parameters or keywords from which one has to be selected. |
| Braces | [ ] | A pair of braces indicate a series of parameters or keywords that can be omitted. |

## Glossary

| Term | Definition |
|---|---|
| Vector Engine (VE) | The NEC original PCIe card for vector processing based on SX architecture. It is connected to x86-64 machine. VE consists of more than one core and shared memory. |
| Vector Host (VH) | The x86-64 architecture machine that VE connected. |
| Vector Island (VI) | The general component unit of a single VH and one or more VEs connected to the VH. |
| Batch Server (BSV) | Resident system process running on a Batch server host to manage entire NQSV system. |
| Job Server (JSV) | Resident system process running on each execution host to manage the execution of jobs. |
| JobManipulator (JM) | JobManipulator is the scheduler function of NQSV.<br>JM manages the computing resources and determines the execution time of jobs. |
| Accounting Server | Accounting server collects and manages account information and manages budgets. |
| Request | A unit of user jobs in the NQSV system. It consists of one or more jobs. Requests are managed by the Batch Server. |
| Job | A job is an execution unit of user job. It is managed by Job Server. |

| | |
|---|---|
| Logical Host | A logical host is a set of logical (virtually) divided resources of an execution host. |
| Queue | It is a mechanism that pools and manages requests submitted to BSV. |
| BMC | Board Management Controller for short. It performs server management based on the Intelligent Platform Management Interface (IPMI). |
| HCA | Host Channel Adapter for short. The PCIe card installed in VH to connect to the IB network. |
| IB | InfiniBand for short. |
| MPI | Abbreviation for Message Passing Interface. MPI is a standard for parallel computing between nodes. |
| NIC | Network Interface Card for short. The hardware to communicate with other node. |

# Contents

## Contents of Figures

## 1. Unit Management

The packages of NQSV is formed with multiple units. In this section, describes about the management of the units. For detail of unit, refer the Linux manual of systemd and systemctl.

NQSV package includes following units.

| Package Name | Unit | |
|---|---|---|
| | Target Unit Name | Service Unit Name and Description |
| NQSV/ResourceManager | nqs-bsv.target | nqs-bsv.service Batch Sever<br>nqs-asv.service Accounting Server<br>nqs-acm.service Accounting Monitor<br>nqs-comd.service Communication Server<br>nqs-nag.service Node Management Agent<br>nqs-btu.service Boot-up Daemon for Redundancy function |
| NQSV/JobServer | nqs-jsv.target | nqs-jsv.service Job Server<br>nqs-lchd.service Launcher demon |
| NQSV/Client | nqs-uag.target | nqs-uag.service User Agent |
| NQSV/JobManipulator | nqs-jmd.target | nqs-jmd.service JobManipulator |

The unit which has .service extension, it called service unit, manages daemon. The unit which has .target extension, it called target unit, controls multiple units.

Each service unit is connected with target units that belongs same package, but it is disabled when installed. So it needs to enable to use service. To enable the service unit, execute following command with root user.

```
# systemctl enable <UnitName>
```

For example, execute following command to enable the nqs-bsv.service and nqs-acm.service.

```
# systemctl enable nqs-bsv.service nqs-acm.service
```

All enabled units can start together by starting target unit. To start the unit, execute following command with root user.

```
# systemctl start <UnitName>
```

For example, execute following command after above example, can start the nqs-bsv.service and nqs-acm.service together.

```
# systemctl start nqs-bsv.target
```

The service unit also start alone. For example, following command starts nqs-bsv.service.

```
# systemctl start nqs-bsv.service
```

To stop the unit, execute following command with root user.

```
# systemctl stop <Unit>
```

On above example, following command stops nqs-bsv.service and nqs-acm.service together these service is enabled with nqs-bsv.target.

```
# systemctl stop nqs-bsv.target
```

The service unit also stop alone. For example, following command stops nqs-bsv.service.

```
# systemctl start nqs-bsv.service
```

The target unit that included in each packages are enabled with multi-user.target. Therefore all units that enabled are automatically start when OS booted. To prevent automatically start with OS boot, disable the target unit with multi-user.target. For example to disable the nqs-bsv.target, execute the following command.

```
# systemctl disable nqs-bsv.target
```

## 2. Batch Server Management

The batch server is the core component of NQSV batch system. It manages requests, queues and communication among each component. (Please refer to [Introduction]Batch server.)

### 2.1. TCP/IP port number of a Batch Server

In NQSV batch system, the batch server acts as a central hub of communication for the batch schedulers, job servers and user commands using a TCP/IP network. Thus, the batch server host (where the batch server resides) and all the other components (client hosts and execution hosts) should be connected over TCP/IP.
The batch server binds to a single TCP port to accept requests from clients. The default port number is 602. A port number specified in /etc/services with the name "nqsv" will be given priority.
And a port number can be specified with nqs_bsvd command's -p option at the batch server start-up.

### 2.2. Files and Directories for Batch Server Setting

The following is the main files and directories needed for batch server operations:

| | |
|---|---|
| /etc/opt/nec/nqsv/nqs_user.map | ... User map file |
| /etc/opt/nec/nqsv/nqs_passwd.def | ... Local account file (for user) |
| /etc/opt/nec/nqsv/nqs_group.def | ... Local account file (for group) |
| /etc/opt/nec/nqsv/nmap/ | ... Machine ID database |
| /etc/opt/nec/nqsv/nqs_bsv.env | ... Configuration file for boot up batch server |
| /opt/nec/nqsv/sbin/nqs_bsvd | ... Batch server |
| /opt/nec/nqsv/sbin/nqs_logd | ... Logging server |
| /opt/nec/nqsv/sbin/nqs_roud | ... Routing server |
| /opt/nec/nqsv/sbin/nqs_stgd | ... Staging server |
| /opt/nec/nqsv/sbin/nqs_comd | ... Communication server |
| /opt/nec/nqsv/sbin/uex_prog/ | ... User EXIT script storage location |
| /opt/nec/nqsv/sbin/stg_prog/ | ... Staging script storage location |
| /opt/nec/nqsv/bin/nmapmgr | ... Machine ID database management command |
| /var/opt/nec/nqsv/ | ... NQSV database |
| /var/opt/nec/nqsv/bsv/ | ... Batch server database |
| /var/opt/nec/nqsv/bsv/private/root/control/ | ... Control file storage location |
| /var/opt/nec/nqsv/bsv/private/root/data/ | ... Shell script storage location |
| /var/opt/nec/nqsv/bsv/private/root/database/queues | ... Batch queue define file |

/var/opt/nec/nqsv/bsv/private/root/database/netqueues
... Network queue define file
/var/opt/nec/nqsv/bsv/private/root/database/params
... Batch server attribute define file
/var/opt/nec/nqsv/bsv/private/root/database_qa/
... Queue access database
/var/opt/nec/nqsv/bsv/private/root/input/ ... Stage-in file storage location
/var/opt/nec/nqsv/bsv/private/root/output/ ... Stage-out file storage location
/var/opt/nec/nqsv/bsv/private/root/failed/ ... Storage location for failed control files
/var/opt/nec/nqsv/bsv/private/root/outfai/ ... Storage location for non-routed result files

- User Map file
  This file defines mapping of remote user names and local user names. This file also defines access privilege to NQSV. (For detail, please refer to 2.9.1. Local User Names and Remote User Names and 2.9.2. User Mapping.)
- Local Account File
  This file enables users who do not have a regular account on the batch server host to access the NQSV batch system. (For the details, refer to 2.9.3. Local Account.)
- Machine ID Database
  This database stores machine IDs added to batch system. (For detail, please refer to 2.5. Machine ID Management
- Batch Server
  This is the main process of the batch server.
- Logging Server
  The logging server processes log output requests received from the batch server and other NQSV servers. Logs are output to /var/opt/nec/nqsv/batch_server_log by default. Fatal errors are output to syslog message. (Facility: LOG_DAEMON, Level: LOG_ERR) Logs will be output to the console if it is not possible to output to syslog.
- Routing Server
  This server routes batch requests in the routing queue and is activated by the batch server at the timing of starting routing process.
- Staging Server
  This server routes request result files. This server routes result files as it is connected to the user agent in the client host. The server is activated by the batch server at the timing of starting staging process.
- Communication server
  This server communicates with the other batch system. This server resides on the batch sever host. (For detail, please refer to 2.10. Communicate with other Batch

server.)

- Machine ID database Management Command

  This command is an administrator command that registers, displays and deletes machine IDs.

  (For detail, refer to 2.5. Machine ID Management.)

- Control File Storage Location

  This directory stores control files for requests. The control file contains all information related to requests.

- Shell Script Storage Location

  This directory stores the shell script specified when a request is submitted. Batch jobs are executed in accordance with information in this script.

- Queue Define File

  This file stores definitions of the queues (batch, interactive and routing queues).

- Network Queue Define File

  This file stores definitions of the network queues.

- Batch Server Attribute Define File

  This file stores various attributes of the batch server.

- Queue Access Database

  This database stores access permission information (enable/disable) to the queue for each user and group.

- Stage-in File Storage Location

  This directory temporarily stores the stage-in files.

- Stage-out File Storage Location

  This directory temporarily stores the stage-out files.

- Storage Location for Failed Control Files

  This directory stores control files of batch requests that can no longer be processed due to a fatal error. As the batch server does not access files under this directory, please delete unnecessary files periodically. (Unnecessary files can be deleted during operation of batch server without any problems.)

- Storage Location for Non-Routed Result Files

  This directory stores result files that could not be staged due to an error or other reason. If necessary, transfer these files manually. As the batch server does not access files under this directory, please delete unnecessary files periodically. (Unnecessary files can be deleted during operation of batch server without any problems.)

## 2.3. Batch Server Setting

### 2.3.1. Log File

The batch server outputs the running logs into a log file. Set this attribute with the **set batch_server log_file** sub-command of the qmgr(1M) command.

The following settings are available for the batch server log file.

- A log file name
- A maximum limit size of log files
- The output level of log files. (More detailed information is output with large level. Please refer to set batch_server log_file sub-command of the qmgr(1M) command.)
- The number of backup files

The default values are as below.

| | |
|---|---|
| A log file name | : /var/opt/nec/nqsv/batch_server_log |
| A maximum limit size of log files | : UNLIMITED (Unlimited) |
| The output level of log files | : 1 |
| The number of backup files | : 3 |

### 2.3.2. Information Interval

The batch server gets the information from the execution hosts and the batch jobs. Specify interval settings using sub-commands of qmgr(1M).

The sub-commands and the default values are as follows.

| Attribute | qmgr (1M) sub-command | Default |
|---|---|---|
| The interval to get load information of execution hosts and HW resource loading information | set batch_server load_interval | 30 seconds |
| The interval to get resource quantity of jobs | set batch_server get_resource_interval | 30 seconds |

### 2.3.3. Submit Limit

It is possible to set a limit to the number of requests which can be submitted to the batch system. The number of requests that can be submitted means the total number of requests existent in the batch system at the same time.

The limits can be specified per system, per group, and per user. Specify the limit to the number of requests using sub-commands of qmgr(1M).

The sub-commands and the default values are as follows.

| Attribute | qmgr (1M) sub-command | Default |
|---|---|---|
| The number of requests which can be | set batch_server submit_limit | 1000 |

| | | |
|---|---|---|
| submitted to the system | | |
| The number of requests which one group can submit to the system | set batch_server group_submit_limit | 0 (unlimited) |
| The number of requests which one user can submit to the system | set batch_server user_submit_limit | 0 (unlimited) |

It is also possible to set submit number limit per individually designated group/user name, please refer to 12. Limit per Group and User.

About submit number limit of queue, please refer to 4.1.2. Batch Queue Configuration, 4.2.2. Interactive Queue Configuration and 4.3.2. Routing Queue Configuration.

### 2.3.4. Routing Limit

It is possible to set a limit to the number of requests in routing queues and network queues that can be executed simultaneously in the entire system. Specify the run limit using sub-commands of qmgr(1M).

The sub-commands and the default values are as follows.

| Attribute | qmgr (1M) sub-command | Default | Max |
|---|---|---|---|
| Routing queue run limit | set batch_server routing_queue run_limit | 100 | 100 |
| Network queue run limit | set batch_server network_queue run_limit | 1000 | 1000 |

[Notes]

Execution of routing will be stopped if the number reaches the limit value set for the entire system even though there is room for the number of simultaneous executions of individual queues. (For the setting of the run limit number of requests each routing queue and network queue, please refer to 4.3.2. Routing Queue Configuration and 4.4.2. Network Queue Configuration.)

### 2.3.5. Routing Retry Interval

Routing Retry Interval is a waiting interval from failure of routing or staging to next retry. The retry interval can be set for routing queues and network queues. Specify the retry interval using sub-commands of qmgr(1M).

The sub-commands and the default values are as follows.

| Attribute | qmgr (1M) sub-command | Default |
|---|---|---|
| Routing queue retry interval | set batch_server routing_queue retry_interval | 300 seconds |
| Network queue retry interval | set batch_server network_queue retry_interval | 300 seconds |

### 2.3.6. Routing Retry Span

Routing Retry Span is a span to repeat routing from the time of first routing process or staging process. The retry interval can be set for routing queues and network queues. Specify the retry span using sub-commands of qmgr(1M).

The sub-commands and the default values are as follows.

| Attribute | qmgr (1M) sub-command | Default |
|---|---|---|
| Routing queue retry span | set batch_server routing_queue retry_span | 259200 seconds (3 days) |
| Network queue retry span | set batch_server network_queue retry_span | 259200 seconds (3 days) |

In a routing queue, in case retry process of transferring requests exceeded the value of retry_span, the corresponding request will be deleted and notifying e-mail will be sent to the owner of the request.

In a network queue, in case a request in STAGING state exceeded the value of retry_span, retry process will not be performed any longer and the request will return to QUEUED state because it is regarded as failure in STAGING. The request returned to QUEUED state will be restarted STAGING by the scheduler.

In case a request in EXITING state exceeded the value of retry_span, retry process will not be performed any longer and transferring process of result files will be cancelled because it is regarded as failure in EXITING. The result files cancelled to transfer will be stored in (/var/opt/nec/nqsv/bsv/private/root/outfai/).

### 2.3.7. Heartbeat Interval

It is possible to set an interval to send heartbeat between the batch server and job servers.

Specify the heartbeat_interval using the set batch_server heartbeat_interval sub-command of qmgr(1M). The default value is 60 seconds.

### 2.3.8. Budget Management Function

NQSV can check the budget overruns cooperating. When a request is owned by a user or group who has exceeded their budget, NQSV refuses to accept their submitted requests and to execute their queued requests.

For detail, please refer to [Accounting & Budget Control].

### 2.3.9. Request Accounting File

It is possible to output accounting information to request account file.

Specify the output setting of the job accounting using sub-commands of qmgr(1M).

The sub-commands and the default values are as follows.

| Attribute | qmgr (1M) sub-command | Default |
|---|---|---|
| Start output of the request accounting information | set batch_server req_account on | OFF |
| Stop output of the request accounting information | set batch_server req_account off | |
| The output file of the request accounting information | set batch_server req_account_file | /var/opt/nec/nqsv/bsv/account/reqacct |
| The output directory of job accounting information | set batch_server jacct_dir | /var/opt/nec/nqsv/acm/jacct |

For detail, please refer to [Accounting & Budget Control].

### 2.3.10. Sub-request of Parametric Request Limit

A batch server restricts the number of the sub-request generated at the same time to each parametric request.

Specify the heartbeat_interval using the set batch_server subrequest_entry_limit sub-command of qmgr(1M). The default value is 100.

### 2.3.11. Getting of License

The license of the NQSV is managed in a separately built license server. Associating with the license server, the batch server checks the number of available licenses based on the license information managed by the license server and determines whether to enable connection of the following licensed products.

- NQSV/JobServer (JSV license)
- NQSV/JobManipulator (JM license)

(1) Configuring the license server

| /opt/nec/aur_license/aur_license.conf |
|---|

Specify the information as follows:

**License_server_host = [host name]**

Specify the host name of the license server.

**License_server_port =[port number]**

Specify the listening port number on the license server.

(2) Getting of License

When starting, the BSV connects to the license server and acquires the licenses to use on itself. The BSV does not start when there is no available JSV license. The BSV starts as long as a JSV license is available, even if there is no available JM license.

- **When multiple batch servers use the same license server.**

When multiple batch servers use the same license server, the number of licenses available for batch server needs to be defined by license type. Make sure that the total number of licenses by license type defined for batch server does not exceed the number of licenses managed on the license server.

Define the number of licenses in the configuration file nqsd.conf stored on the BSV host (/etc/opt/nec/nqsv/nqsd.conf). The format is as follows:

```
<license define name> <number of license>
```

The character strings that can be specified as the license definition name are as follows:

| license define name | Contests |
|---|---|
| jsv_license_use | Number of JSV license |
| jmn_license_use | Number of JM license |

To use ten JSV licenses and ten JM licenses, for example, define as follows:

```
jsv_license_use 10
jmn_license_use 10
```

The BSV does not start if the number of licenses defined in the above configuration file cannot be acquired from the license server.

[Note]

- If the lines that define the number of licenses are missing or the definition is incorrect, the maximum number of licenses managed on the license server will be acquired.
- Be sure to configure the above setting when multiple batch servers use the same license server. The operation is undefined if this setting is not configured.

(3) Waiting for license server preparation

License server must be running when batch server start. Batch server does not start if the license server not running or connection for license server failed. On this case, it needs to start batch server again after the license server preparation is ready.

Batch server can wait by retrying at 1 minute interval to connect to license server. Max number of retrying can be specified for nqsd.conf file. The format is following.

```
retry_license n
```

*n* is the maximum number of retry. It can be specified integer number 0 to 2147483647 and 0 means not wait for license server. If it specified non integer value or this configure not exist, 5 is applied.
After the batch server connects to license server and the batch server gets licenses from it, batch server start operating. Batch server exits if the license server cannot be used after retry *n* times.

### 2.3.12. License Update

To change the number of licenses to be used during BSV operation in such cases below, run the update license subcommand of qmgr to apply the updated value to the BSV.

- When the setting of the number of licenses is changed on the license server
- When the definition of the number of licenses is changed in configuration file nqsd.conf

Run the update license subcommand of qmgr with administrator rights.

```
$ qmgr -Pm
Mgr: update license
Update License
```

### 2.3.13. Referencing the number of available licenses

You can check the maximum number of licenses held by the BSV and the number of licenses currently being occupied by running qstat -Bf.

```
$ qstat -Bf
Batch Server: bhost.example.com
    NQSV Version = R1.00 (linux)
    Batch Server State = Active
    ... skip ...
  Use License :
    License (NQSV/JobServer)      = 36 (Max: 2048)
    License (NQSV/JobManipulator) = 36 (Max: 2048)
```

You can check the detailed information about the licenses held by the BSV by running qstat -B -L -Pm.

```
$ qstat -B -L -Pm
Sysname    Product              Ver   Expiration    Num
---------- -------------------- ----- ----------- -----
VESYS      NQSV/JobServer       V1.0  31-Aug-2019  2048
VESYS      NWSV/JobManipulator  V1.0  01-Sep-2019  2048
```

### 2.3.14. Delayed Writing to DB

When writing the request information managed by the batch server to a file in the DB, the request processing performance can be improved by delayed writing. This function is configured by nqsd.conf file (/etc/opt/nec/nqsv/nqsd.conf). To enable this function, set following line to nqsd.conf and restart the batch server or execute "load nqsd_conf" sub-command of qmgr.

```
no_sync_mode on
```

If this option is not configured, the DB is updated with sync. If this option is configured, the DB is not sync when it updated. Therefore if the batch server aborted by failure, the possibility of request data brake increased. And it maybe cause the data writing delay to the DB area of batch server (/var/opt/nec/nqsv/bsv) according to the type of file system or mount option of it. Please mind this point and be careful to use this function.

### 2.3.15. Accept SIGTERM on job execution

By default, the shell which executes the job ignores SIGTERM. This behavior is set by job server to get the correct exit status of the job by preventing the termination of execution shell which is caused by receiving SIGTERM for resource limit exceeding. But this configuration makes shell script cannot trap SIGTERM.

If user specify the accept sigterm option on qsub command, SIGTERM can be trapped in the job script. Following configuration always makes SIGTERM can be trapped on all jobs. For details for qsub(1) command –accept-sigterm option, please refer to [Operation].

This function is configured in nqsd.conf file (/etc/opt/nec/nqsv/nqsd.conf) as following format. And this configuration is enabled by rebooting batch server or executing "load nqsd_conf" sub-command of qmgr command.

accept_sigterm yes

The default behavior is SIGTERM can be trapped only when accept sigterm option is specified in qsub command. It makes default behavior when this configuration is not specified or specified with wrong format.

[Notes]

- If accept sigterm function is enabled by qsub command option or this batch server configuration, the exit status which recorded in the job account may be different from default behavior (this option is disabled) when the job exited by exceeding the resource limit.

- The accept sigterm option value that is specified on qsub command is directly displayed on the request information of qstat command. This value is not link to the configuration of batch server. Therefore the behavior for accepting SIGTERM of the job is not always regard to the value on qstat command.

### 2.3.16. Setting the Maximum Sequence Number of Request ID

The maximum sequence number of the request ID can be changed. The setting range is 999999 to 99999999, and it is possible to submit from 1 million requests to a maximum of 100 million requests.

This function is enabled by setting the nqsd.conf (/etc/opt/nec/nqsv/nqsd.conf) in the following format. And this configuration is enabled by rebooting batch server or executing "load nqsd_conf" sub-command of qmgr command.

```
rid_seqno_max  <value>
```

Specify <*value*> as an integer in the range of 999999 to 99999999. If not specified, the maximum sequence number of the request ID will be the default value 999999. The maximum sequence number of the request ID is changed to the lower limit (999999) if you specify a value less than the lower limit, and the maximum sequence number of the request ID is changed to the upper limit (99999999) if you specify a value that exceeds the upper limit.

### 2.3.17. Setting the Sequence Number of the Next Submitted Request

The sequence number assigned to the next submitted request can be set. The setting range is from 0 to the maximum value of the sequence number. For the maximum value of the sequence number, refer to "2.3.16 Setting the Maximum Sequence Number of Request ID".

This function is enabled by setting the seqno.conf (/var/opt/nec/nqsv/seqno.conf) in the following format. And this configuration is enabled by rebooting batch server or executing "load nqsd_conf" sub-command of qmgr command.

```
<value>
```

Specify <value> as an integer in the range of 0 to the maximum sequence number of the request ID. If a value outside the range is specified, the specified value is ignored. Also, the specified value is assigned. If a request assigned with the specified value already exists, the specified value is ignored and an available value greater than that value is assigned.

The seqno.conf is not installed by the NQSV/ResourceManager package. If you want to change the settings, create seqno.conf and place it under /var/opt /nec /nqsv.

In order to avoid accidentally applying the past settings when BSV is rebooted due to a failure, etc., the seqno.conf is renamed to “ seqno.conf.accept” after being applied.

### 2.3.18. User Notification Script Settings.

The node health check function can notify the request owner via e-mail or other means when an abnormality is detected. The notification is done by a shell script. The script contents and script directory can be customized freely by the user.
This script should be set in the nqsd.conf file (/etc/opt/nec/nqsv/nqsd.conf) in the following format. After that, restart the batch server or execute the "load nqsd_conf" subcommand of qmgr to activate it.

```
notify_script_path <script_path>
```

<script_path> should be the path to the user notification script. If the above is not specified, the default notification script below will be used.

/opt/nec/nqsv/sbin/notify_prog/nqsv_notify.sh

For information on how to create scripts and the default script behavior, please refer to Section 19.3 Node Health Check Function.

### 2.3.19. Memory management with memory cgroup

Memory management using cgroups from the Linux kernel is available. You can get memory usage and limit resources for each job accurately. Enable and disable this feature for the batch server. It is not possible to change the memory management method of this function for each queue or execution host.

To use this function, please write the settings in the following format in the nqsd.conf file (/etc/opt/nec/nqsv/nqsd.conf). It is then enabled by restarting the batch server or by re-loading the configuration file by running the "load nqsd_conf" subcommand in qmgr.

Settings are made using three types of configuration parameters.

1. enable_memory_cgroup

If this feature is enabled, memory cgroup will be used to manage memory resources. Process ID management for each job is also changed to use cgroup. Please write the settings in the following format in the nqsd.conf file. Without this specification, this feature is disabled (the default).

```
enable_memory_cgroup on
```

2. memory_cgroup_excluding_cache

If memory cgroup is used for memory management, the file cache is also included in the memory usage. To exclude the file cache from memory usage, describe the settings in the following format. If this is not specified, this function is disabled (the default).

```
memory_cgroup_excluding_cache on
```

3. enable_cgroup_process_list

Process ID management for jobs uses cgroups. Please write the settings in the following format in the nqsd.conf file. Without this specification, this feature is disabled (the default). However, if this is not specified and enable_memory_cgroup is "on", this feature is enabled.

```
enable_cgroup_process_list on
```

These configuration parameter combinations are:

*1: enable_memory_cgroup

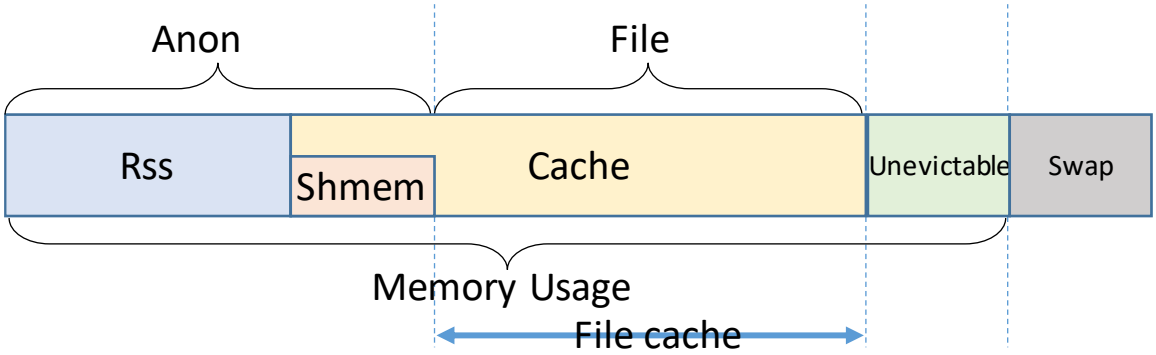
*2: memory_cgroup_excluding_cache

*3: enable_cgroup_process_list

| Config parameters | | | Function performed | | |
|---|---|---|---|---|---|
| *1: memory | *2: cache | *3: process | *1: memory | *2: cache | *3: process |
| on | on | - | yes | yes | - |
| on | off | - | yes | no | - |
| on | *not set* | - | yes | no | - |
| on | - | on | yes | - | yes |
| on | - | off | yes | - | no |
| on | - | *not set* | yes | - | yes |
| off | *don't care* | on | no | no | yes |
| off | *don't care* | off | no | no | no |
| off | *don't care* | *not set* | no | no | no |
| *not set* | *don't care* | on | no | no | yes |
| *not set* | *don't care* | off | no | no | no |
| *not set* | *don't care* | *not set* | no | no | no |

**[Notes]**

- The file cache that is excluded by specifying “memory_cgroup_excluding_cache on” is cache minus Shmem.




- Memory usages of the following functions include file cache when "memory_cgroup_excluding_cache off" is specified..
  - Memory usages in the basic and detail information of the request and job by qstat.
  - Memory usages in the request and job account.
  - Charge for actually used memory usages.

**Target memory resource limits**

The following maximum value of the limit for memory resources are covered:

| option | meaning |
|---|---|
| -l --memsz_job | Memory size per job(logical host) |
| --memsz-lhost | Memory size per job(logical host) |

Virtual memory size limit (--vmemsz-lhost) and warning value of limit are controlled as before.

**Behavior when memory resource limit is exceeded**

NQSV manages the flag of memory.oom_control as oom_kill_disable 1. If the memory resource limit is exceeded, the process is not KILLed by OOM Killer and is in a stopped state. NQSV checks the process stopped by OOM and kills the job by SIGKILL sending.

**Browsing memory usage with qstat -Jf**

If memory management by cgroup is enabled, qstat -Jf displays memory usage details.

```
        Resources Information:
          Memory     = 580.878906MB
          Memory Cgroup Resources = {
            Memory Usage       = 580.906250MB
            Max Memory Usage   = 606.296875MB
            Cache              = 529.683594MB
            Rss                = 51.222656MB
            Shmem              = 529.656250MB
            Unevictable        = 0.000000B
```

```
        Swap                = 25.343750MB
    }
```

### 2.3.20. Tuning the Stage-Out Processing Buffer of the Request Result File

This function tunes the buffer size used when transferring request result files from the batch server to the user agent using the internal staging method.

Specify the appropriate file buffer size for your file system type and mount options. In addition, specify the sync mode (whether or not to sync), which is a write guarantee.

Set this function to the nqsd.conf file (/etc/opt/nec/nqsv/nqsd.conf) in the following format. After that, restart the batch server or reload the configuration file using the qmgr "load nqsd_conf" subcommand.

| Item | Meaning |
|---|---|
| file_transfer_buffer_size | Specify the file buffer size, in bytes, appropriate for your file system type and mount options.<br>The valid range is 1024~10485760 (1Kbyte~10Mbytes). |
| file_transfer_sync_mode | Specifies whether write operations are synchronized when the file is closed.<br>It is "on" to synchronize, "off" if not to synchronize. |

Setting example: When file buffer size is 8Kbytes and Sync mode works enable

```
    file_transfer_buffer_size 8192
    file_transfer_sync_mode on
```

To specify the buffer size file_transfer_buffer_size enter the buffer size in byte units. The valid range is 1024~10485760 (1Kbyte~10Mbytes).

To specify the sync mode, enter file_transfer_sync_mode followed by "on" or"off".

If this setting is not performed, the stage-out processing buffer of the request result file will be operated at 1048576 (1Mbyte). Sync mode is "on" (with sync before closing file).

If the value is not specified or out of the setting range, the operation will be the same as if the above setting is not performed.

Processing delay may occur depending on your environment (number of concurrent requests, file system type and mount options). Therefore, when making this setting, please be careful to determine the value.

### 2.3.21. Change the behavior of submit request number limitation exceeding during request routing

Request forwarding from the routing queue to the batch queue is affected by the submit limit of requests set for the queue. The behavior of when the submit limit of requests exceeded is changed.

The sub-commands and the default values for execution queue are as follows.

| Attribute | qmgr(1M) sub-command | Default |
|---|---|---|
| The number of requests that can be submitted to a queue | set execution_queue submit_limit | 0 (unlimited) |
| The number of requests that one group can submit to a queue | set execution_queue group_submit_limit | 0 (unlimited) |
| The number of requests that one user can submit to a queue | set execution_queue user_submit_limit | 0 (unlimited) |

These maximum numbers of submit request are managed according to their respective limit values. For example, if a user submits a request to the routing queue and the destination of batch queue exceeds the user's limit number of submit request, the destination of batch queue temporarily becomes unavailable for request forwarding, preventing other users' request forwarding also.

To change this behavior, we allow requests from other groups or users to be forwarded to the routing queue of a certain group or user even when the submit request limit of the destination queue has been exceeded. Please set to nqsd.conf file (/etc/opt/nec/nqsv/nqsd.conf) with the following format.

```
change_routing_retry_behavior on
```

If this is not specified, this function is disabled (the default). After that, it can be enabled by restarting the batch server or reloading the configuration file by using the "load nqsd_conf" subcommand of qmgr(1M).

The behavior about the submit limit of queue (submit_limit) or other limitation of a queue are not changed.

### 2.4. Batch Server Status Check

To see the batch server status, execute the qstat(1) command with **-B** option.

```
$ qstat -B
BatchServer      MachineID UMAX GMAX RRL NRL   TOT ARR WAI QUE RUN EXT HLD SUS Status
---------------- --------- ------------------- ---------------------------------- ------
bsv.example.com        100    0    0 100 1000    0   0   0   0   0   0   0   0 Active
```

Execute the qstat(1) command with **-B, -f** option to get the detail information of the batch server.

```
$ qstat -B -f
Batch Server: bsv.example.com
    NQSV Version = R1.00 (linux)    Batch Server State = Active
    Batch Server Machine ID = 100
    Logfile Path = /var/opt/nec/nqsv/batch_server_log
    Logfile Level = 2
    Logfile Save Count = 3
    Logfile MAX Size = 512.0MB
    Output Account = OFF
    Accounting Server Host Name = localhost
    Accounting Server Port Number = 4595
    Jacct directory = /var/opt/nec/nqsv/acm/jacct
    Budget function              = OFF
    Request Accounting = OFF
    Request Accounting File Path = /var/opt/nec/nqsv/bsv/account/reqacct
    Reservation Accounting       = OFF
    Reservation Accounting File  = /var/opt/nec/nqsv/acm/rsvacct/rsvacct
    Specify Group for Request = ON
    Allow Absolute Exepath = OFF
    Auto Delete Failed Request = ON
    Total Request = 0
    Arriving Request = 0
    Waiting Request = 0
    Queued Request = 0
    Pre-running Request = 0
    Running Request = 0
    Post-running Request = 0
    Exiting Request = 0
    Held Request = 0
    Holding Request = 0
    Restarting Request = 0
    Suspending Request = 0
    Suspended Request = 0
    Resuming Request = 0
    Migrating Request = 0
    Transiting Request = 0
    Staging Request = 0
    Checkpointing Request = 0
    Heart Beat Interval = 60S
```

```
    Load Interval = 30S
    Get Resource Interval = 30S
    Max Subrequest Entry Limit = 100
    Max Run Limit of Routing Queue = 100
    Retry Interval of Routing Queue = 300S
    Retry Span of Routing Queue = 259200S
    Max Run Limit of Network Queue = 1000
    Retry Interval of Network Queue = 300S
    Retry Span of Network Queue = 259200S
    Submit Number Limitation Value = 1000
    Submit User Number Limitation Value = UNLIMITED
    Submit Group Number Limitation Value = UNLIMITED
  Use License :
    License (NQSV/JobServer) = 2 (Max: 16384)
    License (NQSV/JobManipulator) = 1 (Max: 16384)
```

It is possible to display information by selecting and customizing each item output by qstat(1) command.

Please specify items to be displayed with **-F** option of qstat(1) command.

```
$ qstat -B -F bsvhost,mid,bstt
BatchServer      MachineID Status
---------------- ---------- ------
bsv.example.com        147 Active
```

The items that can be specified are as follows.

| Item | Description | Item Name |
|---|---|---|
| bsvhost | Batch server host name | BatchServer |
| mid | Machine ID | MachineID |
| umax | Request submit limit per user | UMAX |
| gmax | Request submit limit pre group | GMAX |
| rrlm | Routing queue run limit | RRL |
| nrlm | Network queue run limit | NRL |
| tot | Number of batch requests managed by the batch server | TOT |
| arr | Number of batch requests for each state | ARR |
| wai | | WAI |
| gqd | | GQD |
| que | | QUE |
| run | | RUN |
| ext | | EXT |
| hld | | HLD |
| sus | | SUS |

| | | |
|---|---|---|
| fwd | | FWD |
| bstt | Batch server status | Status |

(For details for how to use **qstat**(1) command **-F** option, please refer to [Operation] Customizing Information.)

It is also possible to sort information by any item using -o option or -O option of qstat(1) command. The items for sorting are same as the items shown above and more than one items can be specified with delimited by comma ",". (For details for how to use this option, please refer to [Operation] Sorting Information.) Sometimes there are cases that information such as execution host name is cut off in the basic information display of qstat(1) command (without -f option) because the viewable number of character is fixed. By using -l option, it is possible to show all information without being cut. (For details for how to use this option, please refer to [Operation] Sorting Information.)

## 2.5. Machine ID Management

### 2.5.1. Machine ID

A machine ID is an integer value of 32 bits and is used to identify batch server. On NQSV, a machine ID is allocated to one batch server host. It is not necessary to set machine IDs for the client hosts and execution hosts.

### 2.5.2. Machine ID Setting

Machine IDs are allocated by nmapmgr(1M). Use subcommand **add mid** of nmapmgr by root privilege when allocating Machine ID 100 to a host whose official name is host0.example.com.

```
# nmapmgr
NMAPMGR>: add mid 10 host0.example.com
```

One machine ID is allocated to each host. Machine IDs do not have to be allocated to individual network interfaces of each host.

### 2.5.3. Alias Hostname Setting

When the host has another host name besides an official name, because the host has several network interfaces, it is possible to set it as an alias by **add name** subcommand of nmapmgr(1M) command.

```
# nmapmgr
NMAPMGR>: add name host1.example.com 10
```

By setting an alias name, the host name is always converted to the same machine ID regardless of which host name is used to access. Thus it is possible to uniquely identify the batch system.

### 2.6. Initializing Batch Server Database

A dedicated database (/var/opt/nec/nqsv/bsv/) must be constructed in order to execute the batch server. Initialize the database after package installation or when the database is destroyed. Initialize the database by activating the batch server (/opt/nec/nqsv/sbin/nqs_bsvd) by **-i** option by root privilege.

```
# /opt/nec/nqsv/sbin/nqs_bsvd -i
BatchServer database was initialized.
```

If a database already exists, a message for confirmation will be displayed, prompting an input.

```
# /opt/nec/nqsv/sbin/nqs_bsvd -i
BatchServer database is already exists. May I delete it (yes/no/save)?
```

If "yes" is input:

The existing database will be deleted and a new database will be created.

If "no" is input:

Database initializing will be stopped.

If "save" is input:

The existing database will be saved as "/var/opt/nec/nqsv/bsv.old" and a new database will be created.

### 2.7. Batch Server Activation

Execute the following by root privilege when manually activating the batch server:

```
# systemctl start nqs-bsv.service
```

To start batch server automatically when batch server host booted, enable nqs-bsv service.

```
# systemctl enable nqs-bsv.service
```

To start all service units which enabled together, execute following command.

```
# systemctl start nqs-bsv.target
```

[Notes]

Owner unknown request in the batch server is deleted by batch server when it rebooted. If start nqs_bsvd with **-u** option, owner unknown request is not deleted and batch server aborts. This option can save owner unknown request when the batch server rebooted while authentication service temporary disabled. But if the request's owner account has lapsed while batch server is not start, the batch server can't start.

On this situation, batch server can start by deleting unknown owner request without **-u** option.

### 2.8. Batch Server Stop

The batch server is stopped when the batch server host is shut down. Execute the following by root privilege or execute shutdown sub-command of qmgr(1M) when manually stopping the batch server:

```
# systemctl stop nqs-bsv.service
```

Following command can stop all service unit together which enabled.

```
# systemctl stop nqs-bsv.target
```

If the batch server is stopped, the each job server connected will detect the down of the batch server and will shift to the mode that will periodically attempt reconnection and the control of the batch jobs will be continued as is. It is possible to continue execution of batch jobs with this feature even if the batch server is down during execution of batch jobs.
If a batch job is completed during shutdown of the batch server, the job server records the stores end status in its own database (For details of job server database, please refer to 3.1. Files and Directories for Execution Host Setting) until the job server is reconnected to the batch server.
When the batch server is restarted, each job server reconnects to the batch server and reports the status of the batch jobs under its control to the batch server. The batch server reconstructs job management data based on this information and returns to the normal operation status.

### 2.9. User Management

#### 2.9.1. Local User Names and Remote User Names

Local user names are user names in the batch server host. They are used in the NQSV to check access privileges and to identify the request owners.
Remote user names are user names that execute user commands in the client host and executes the batch jobs in the execution host.
If local user names and remote user names are different, the relationship between these user names must be defined. The NQSV implements the **user mapping function** to define the relationship between local and remote user names and implements the **local account function** to define user and group account information. Thus it enables to define relationship between different user names flexibly.

#### 2.9.2. User Mapping

The user mapping function is a function to relate local and remote user names according to the definition in the user map file (/etc/opt/nec/nqsv/nqs_user.map) on the batch server host. When

/etc/opt/nec/nqsv/nqs_user.map file is renewed, it is contents will be reflected immediately to a batch server.

User mapping is performed from remote user names to local user names (called RL mapping) and from local user names to remote user names (called LR mapping). Mapping is processed at the following timings:

RL Mapping

When the scheduler connects to the batch server:

Mapping is performed from the remote user name that activated the scheduler to a local user name.

When the user/manager command connects to the batch server:

Mapping is performed from remote user name that activated the command to a local user name.

LR Mapping

When a job server creates a job, initiated by the batch server:

Mapping is performed from the local user name (request owner) to a remote user name (job owner on the execution host)

When a job migrates from one job server to another, initiated by the batch server:

Mapping is performed from the local user name (request owner) to a remote user name (job owner on the execution host)

When a batch request result file is staged from the batch server to the client host:

Mapping is performed from local user name (request owner) to a remote user name (job owner on the execution host)

**Format of User Map File**

A sample of the user map file is shown below.

```
#
# NQSV User Mapping File.
#

DefaultPrivilege PRIV_SPU

#-----------------------------------------------------------------------------
# Privilege LocalUser RemoteUser (s)
#-----------------------------------------------------------------------------

PRIV_SCH root root:127.0.0.1/32
PRIV_NON root root:0.0.0.0/0
PRIV_MGR nqsmgr user00:192.168.77.0/24 user01:172.16.128.0/25
PRIV_OPE nqsope user00:192.168.10.0/24
```

The description of the user map file is as follows. The line numbers correspond to the above

sample file.

- Characters from "#" to line end and blank lines are ignored.
- DefaultPrivilege of the 5th line specifies the default privilege in case no local/remote user mapping exists.
- Local user names (2nd column) and remote user names (3rd column) are mapped in the other lines. (From lines 11 to 14)
- The priority for lines is top down (in other words, the preceding line has the priority) when multiple entries for the same local or remote users exist in the user map file.
- If several remote users are specified in 3rd column and later, the left most remote user matched will have the priority.

A detailed explanation of each column of the line to associate a local user name with a remote user name is as follows.

- 1st column

  The highest limit of the access privilege to remote users matching this line. One of the following keywords can be specified.

| Keyword | Type | Description |
|---|---|---|
| PRIV_SCH | Scheduler privilege | The highest privilege for accessing all functions of the batch server. Mainly used by the scheduler. |
| PRIV_MGR | Manager Privilege | The privilege for managers in order to create and delete queues, delete batch requests of other users, shutdown the batch server and other functions. It is the highest privilege next to PRIV_SCH. |
| PRIV_OPE | Operator Privilege | The privilege for operators in order to stop and restart queues, hold/release batch requests of other users, change attribute values and other functions. |
| PRIV_GMGR | Group Manager Privilege | The privilege for group mangers.<br>The group manager can delete, hold/release, change attribute values and operate other functions about the requests when the groups of the requests are under the group manager's management. |
| PRIV_SPU | Special User Privilege | The privilege for special users, which can refer to the attribute value of the batch request and the job of other users in addition |

| | | |
|---|---|---|
| | | to the general user privilege. |
| PRIV_USR | General User privilege | The privilege for general users, is the lowest privilege. It is limited to operations for batch requests owned by the user, for searching for attribute values of allowed queues and servers and other functions. |
| PRIV_NON | No Privilege | No privilege to access the NQSV batch system. This is specified for remote users to deny access. |

- 2nd column

  A local user name is specified.

  - In RL mapping, users matching remote user specification in the 3rd column and after are mapped in this local user name.
  - In LR mapping, remote user names matching this local user names will be retrieved from remote users specified in the 3rd column and after.

- 3rd column and after

  A remote user list is specified. Multiple remote users are specified with delimited by spaces.

  - In RL mapping, remote user names will be retrieved from this list.
  - In LR mapping, a local user name specified in the 2nd column will be mapped to a remote user name matching in this list.

  The syntax of the remote user list is shown below:

  ```
  <User name>:< IP address>/<Bit mask> [<User name>:< IP address>/< Bit mask> ...]
  ```

  The IP address of the target remote host and IP address specified by <IP address> are checked from the left for the number of bits specified by <Bit mask>. If both addresses match, <User name> will be used as a remote user name.

- next line: (only for PRIV_GMGR)

  In the next line of each PRIV_GMGR (Group Manager Privilege) settings, which groups are under the group manager's management can be specified after leading ":". The multiple groups can be specified, delimited by spaces. If no next line (first character is ":"), the group manager manages no group.

  The syntax of the Group Manager Privilege is shown below by 2 lines :

  ```
  PRIV_GMGR <local user name> <remote user list>
  ```

```
:<group name> [<group name> ...]
```

If a matching entry cannot be found in the user map file, the remote and local user names will be treated as identical and access privilege will be set to the default privilege. The default privilege can be specified by DefaultPrivilege. When DefaultPrivilege is not set, the default privilege is PRIV_USR.

**An example of User Map file**

Actual user mapping is described using the sample shown before as an example.

The first two lines define the remote user name "root."

```
PRIV_SCH root root:127.0.0.1/32
PRIV_NON root root:0.0.0.0/0
```

- For RL mapping

  By specification in line 1, access by "root" in the batch server host is mapped to the local user name "root" and access privilege of PRIV_SCH will be enabled. (127.0.0.1/32 matches only loopback IP Address 127.0.0.1)

  By specification in line 2, access by "root" from all remote hosts (all IP addresses are matched if bit mask is 0) is prohibited. Note that accessing from a local host is not prohibited by specification in line 1 that has higher priority.

- For LR mapping

  By specification in line 1, job execution on the local host is done as root. In result file staging to a local host, the owner of created files is set as root. Job execution and result file staging on the hosts other than local host are prohibited by specification in line 2.

The following two lines define remote user names "user00" and "user01".

```
PRIV_MGR nqsmgr user00:192.168.77.0/24 user01:172.16.128.0/25
PRIV_OPE nqsope user00:192.168.10.0/24
```

- For RL mapping

  By specification in line 1, user "user00" in a remote host whose IP address matches 192.168.77.0/24 and user "user01" in a remote host whose IP address matches 172.16.128.0/25 are mapped to the local user name "nqsmgr" with access privilege of PRIV_MGR. However, the user will be mapped to the local user name "nqsope" and only access privilege PRIV_OPE will be authorized if user "user00" accesses from 192.168.10.0/24 in a remote host by specification in line 2.

Accessing by user "user00" from a remote host that does not exist in the remote user list will be mapped to the local user name "user00" and access privilege of PRIV_USR. It is same in case of user "user01".

- For LR mapping

  When a batch request owned by "nqsmgr" is executed, the remote user (job owner) will be "user00" when a batch job is created on an execution host that matches 192.168.77.0/24 and the remote user (job owner) will be "user01" on the execution host matching 172.16.128.0/25. In other execution hosts, the remote user (job owner) will be "nqsmgr".

### 2.9.3. Local Account

Use the local account function to define the original user/group account information.

The local account function enables user who does not have a regular account on the batch server host to access to the NQSV batch system by preparing NQSV dedicated account database.

The local user name on a batch server host must be registered in the system account database provided by operating system (/etc/passwd file or NIS), but there may be the case the system administrator cannot prepare an account for general user due to security reasons. In such cases, please use the local account function.

The local account is defined by /etc/opt/nec/nqsv/nqs_passwd.def file and /etc/opt/nec/nqsv/nqs_group.def file on the batch server host. The format of these files is quite similar to the /etc/passwd file or /etc/group file. You can use the /etc/passwd, /etc/group or the NIS map file for nqs_passwd.def or nqs_group.def file without needing any modifications. /etc/opt/nec/nqsv/nqs_passwd.def and /etc/opt/nec/nqsv/nqs_group.def are referred to only when an entry of a target user was not found in an account database of system standards.

- nqs_passwd.def

  nqs_passwd.def is a file to define user information. Each line is interpreted as a record for each user and contains at least 4 columns separated by ":". If a line has more than 4 columns, every line from the 5th column onwards is ignored.

| Column | Description |
|---|---|
| 1. | User name |
| 2. | (Unused) |
| 3. | User ID |
| 4. | Group ID |

- nqs_group.def

  nqs_group.def is a file defining group information. Each line is interpreted as a record for each group and contains at least 3 columns separated by ":". If a line has more than 3 columns, every line from the 4th column onwards is ignored.

| Column | Description |
|---|---|
| 1. | Group name |
| 2. | (Unused) |
| 3. | Group ID |

If nqs_passwd.def or nqs_group.def are modified the batch server reloads these files automatically.

## 2.10. Communicate with other Batch server

It is possible to forward a request between batch servers through a communication server in NQSV.

### 2.10.1. Server Setting

To forward a request between batch servers, machine ID need to be registered, and activate a communication server. In addition, at starting user agent, specify batch server.

1. Machine ID registration

   Register the Machine ID for the batch server to communicate as well as the machine ID. For the self-batch server on batch server host. About a registration method of the machine ID, please refer to 2.5. Machine ID Management. )

   It is necessary to assign the unique machine ID.

2. Activate Communication Server

   The communication sever can be activated on boot automatically. Execute the following by root privilege when manually activating the communication server:

```
# systemctl enable nqs-comd.service
# systemctl restart nqs-bsv.target
```

3. Activate User Agent

   On the client host which is used to submit the request, set all batch server host name to BSV_HOSTS variable which exist in the setting file of user agent service (/etc/opt/nec/nqsv/nqs_uag.env).

### 2.10.2. Routing Request Setting

Setup a routing queue to forward a request, and set a queue name on the forwarding batch server by the form of "Queue Name@Batch Server Host Name".

About a setting method of generation of a routing queue, please refer to 4.3.1.Create Routing Queue and 4.3.2.Routing Queue Configuration.)

# 3. Execution Host Management

## 3.1. Files and Directories for Execution Host Setting

The following files and directories are needed for execution host operations:

| | |
|---|---|
| /etc/opt/nec/nqsv/nqs_jsv.env | ... Configuration file for boot up job server |
| /usr/opt/nec/nqsv/sbin/nqs_shpd | ... Job server |
| /usr/opt/nec/nqsv/sbin/nqs_lchd | ... Launcher demon |
| /var/opt/nec/nqsv/jsv/ | ... Job server database |
| /var/opt/nec/nqsv/jsv/<jsvno>/ | ... Job server individual database (< jsvno> is a Job server number) |
| /var/opt/nec/nqsv/jsv/<jsvno>/nqs_shpd.pid | ... Job server PID file |
| /var/opt/nec/nqsv/jsv/<jsvno>/<x.y.z>/ | ... Job database (<x.y.z> is a job ID) |
| /var/opt/nec/nqsv/jsv/jobfile/<x.y.z>/ | ... Job file storage location |
| /var/opt/nec/nqsv/jsv/jobfile/<x.y.z>/stdout | ... Standard output file |
| /var/opt/nec/nqsv/jsv/jobfile/<x.y.z>/stderr | ... Standard error output file |
| /var/opt/nec/nqsv/jsv/jobfile/<x.y.z>/stgfile/ | ... Staging file storage location |

- Job server

  This is the main process of the job server.
- Launcher demon

  This is the daemon to perform the remote control of the job server. The job server can be started and stopped by qmgr(1M) command on the execution hosts on which the launcher daemon resides. (For details, please refer to 3.5.1. Startup by qmgr (1M).)
- Job Server Database

  This is a database for Job server. < jsvno> is a job server number with four digits (left 0 padding).
- Job Server PID File

  This file stores process IDs of the job servers in ASCII format.
- Job Database

  This directory stores information of the jobs under control of the job server. This directory is created when a parent request is changed to STAGING state and is deleted when the request is in EXITING state.
- Job File Storage Location

  This directory stores the job files (that are the files accessed by the job). This place is created when a parent request is changed to STAGING state and is deleted when the request is in EXITING state.
- Standard Output File

  Standard output of processes in the batch job is redirected to this file.

- Standard Error Output File

  Standard error output of processes in the batch job is redirected to this file.
- Staging File Storage Location

  This directory stores the stage-in and stage-out files for the batch job.

### 3.2. Execution Host Registration

The execution host used for execution of a request in a NQSV system is to be registered in batch server.

It is necessary to start one job server to each execution host to manage and execute jobs. It is necessary to assign the job server number to each job server for batch server to distinguish job server uniquely. The job server number needs to become unique in each job server and an integer number from 0 to 10239.

The job server can be started on the execution host by the attach execution_host sub-command of qmgr(1M) by root privilege, specifying host name of the execution host and the job server number.

```
$ qmgr -Pm
Mgr: attach execution_host host = ehost001 job_server_id = 101
Attach Execution_Host (ehost001 : JSVID = 101).
Mgr:
```

It is necessary to register all execution hosts. It is possible to lump together by making the host name of the execution host registered and a list file of the job server number, and then designate the file to register whole execution hosts.

A form of a list file of execution host divides the host name and the job server number by a space character as follows.

```
ehost100 100
ehost101 101
ehost102 102
      :
```

Designate a pathname of this list file and carry out the "attach execution_host" sub-command of the qmgr (1M) command as follows.

```
$ qmgr -Pm
Mgr: attach execution_host file = /tmp/hlist_file
Attach Execution_Host (ehost100 : JSVID = 100).
Attach Execution_Host (ehost101 : JSVID = 101).
Attach Execution_Host (ehost102 : JSVID = 102).
      :
Attach Execution_Host (file = /tmp/hlist_file).
```

When a job server is started with the job server number directly on the execution host,

execution host is registered automatically on the connected batch server. It is explained by the next chapter. It is not possible to register plural job servers on one execution host. In addition, it is not possible to assign the same job server number on different execution hosts.

## 3.3. Node Group

It is possible to handle more than one execution host as one group, and then start job server or bind job server with queue by the group unit. A group of this execution host is called a node group.

### 3.3.1. Create Node Group

Create a node group using the create node_group sub-command of qmgr(1M) command by root privilege. The example When a node group's name is the "ng1", the example is below.

```
$ qmgr -Pm
Mgr: create node_group = ng1
Node group nag created.
```

When a node group is generated, it does not have any execution nodes.
There are three types of group. One is a common node group type, which has more than one execution nodes. Another is a network topology group type, which is used to schedule with network topology. The other is a cloud type, which is bursting jobs to computing resources in the cloud.

| Type | Description |
|---|---|
| common | A usual node group to handle more than one execution hosts as a group |
| nw_topo | A node group to schedule with network topology that define the group to handle more than one execution hosts exists on the same network switch. |
| cloud | A node group to bursting jobs to computing resources in the cloud. |

When creating a node group without type designation like an example of above-mentioned, a node group of common-type is generated. When just mentioning "node group", it means a common node group type. At grouping for the operation more than one execution host, please use a node group of common-type.
At creating of a node group with designation of nw_topo type, a node group of network topology type is generated.

```
$ qmgr -Pm
Mgr: create node_group = nwtplg type = nw_topo comment = "NW topology"
Node group nwtplg created.
```

About a nw_topo group type, please refer to [JobManipulator].

When creating the cloud type of node group, specify "cloud" as the node group type.

```
$ qmgr -Pm
Mgr: create node_group = clng type = cloud comment = "Cloud bursting"
Node_group clng created.
```

About a details of cloud-bursting feature, please refer to [JobManipulator].

### 3.3.2. Setting of Comment of Node Group

It is possible to add a comment as the attribute of the node group. When creating a node_group, it is possible to set a comment. In addition, it is possible to set or change a comment by the "set node_group comment" sub-command of qmgr.

```
$ qmgr -Pm
Mgr: set node_group comment = "My nodegroup" ng1
Set comment to Node_group (ng1).
```

### 3.3.3. Addition of Execution Host to Node Group

The "edit node_group add" sub-command of the qmgr (1M) command is used to add execution hosts to a node group. At this time, the job server number of the added execution host is designated. When adding the job server 100 to the node group "ng1", the sample is below.

```
$ qmgr -Pm
Mgr: edit node_group add job_server_id = 100 ng1
Add Job_Server to Node_group (ng1).
```

It is possible to designate more than one job server number, when adding more than one execution hosts to a node group.

```
$ qmgr -Pm
Mgr: edit node_group add job_server_id = (101,103,107)  ng1
Add Job_Server to Node_group (ng1).
```

When the job server number is a serial number, it is possible to add execution hosts to a node group by range specification of the job server number as follows.

```
$ qmgr -Pm
Mgr: edit node_group add job_server_id = 110-120 ng1
Add Job_Server to Node_group (ng1).
```

When there is a node group that has more than one registered execution hosts, it is possible to add the same execution hosts to a different node group by designating the node group name.

```
$ qmgr -Pm
Mgr: edit node_group add node_group = ng1 ng2
Add node_group to Node_group (ng2).
```

The whole execution host that belongs to node group ng1 is added to node group ng2 by an

example above-mentioned. If the execution host that belongs to ng1 is changed after that, the execution host added to ng2 is not influenced at all.

### 3.3.4. Deletion of Execution Host from Node Group

The "edit node_group delete" sub-command of the qmgr (1M) command is used to delete execution hosts from a node group. At this time, the job server number of the deleted execution host is designated. When deleting the job server 100 from the node group "ng1", the sample is below.

```
$ qmgr -Pm
Mgr: edit node_group delete job_server_id = 100 ng1
Delete Job_Server from Node_group (ng1).
```

It is possible to designate more than one job server number, when deleting more than one execution hosts from a node group. When the job server number is a serial number, it is possible to delete execution hosts from a node group by range specification of the job server number. When there is a node group that has more than one registered execution hosts, it is possible to delete the same execution hosts to a different node group by designating the node group name.

```
$ qmgr -Pm
Mgr: edit node_group delete job_server_id = (101,103,107)  ng1
Delete Job_Server from Node_group (ng1).
$ qmgr -Pm
Mgr: edit node_group delete job_server_id = 110-120 ng1
Delete Job_Server from Node_group (ng1).
$ qmgr -Pm
Mgr: edit node_group delete node_group = ng1 ng2
Delete Job_Server from Node_group (ng1).
```

### 3.3.5. Deletion of Node Group

The "delete node_group" sub-command of the qmgr (1M) command by root privilege is used to delete a node group. At this time, the node group needs to be unbound from any queue.

```
$ qmgr -Pm
Mgr: delete node_group = ng1
Node_group ng1 deleted.
```

About binding of a node group and a queue, please refer to 3.6. Binding Job Server and Queue.

## 3.4. Execution Node Group Information

It is possible to refer the state of the node group using qstat (1) command with -G option.

```
$ qstat -G
NodeGroup       Type      BatchServer     Comment          JSVs BindQueue
--------------- --------- --------------- ---------------- ---- --------------------
ng1             common    bsv1.example.co (none)              1 bq1
```

```
nwtg            nw_topo   bsv1.example.co NW topology         0 (none)

cl_ng1          cloud     bsv1.example.co cloud               1 cloud_bq
```

In addition, it is possible to get the detail information of each node group using -G, -f option.

```
$ qstat -G -f
Node Group: ng1
    Type = common
    Comment = (none)
    Bind Queue list = {
      bq1
    }
    Job Server number list = {
        100.
    }

Node Group: nwtplg_grp
    Type = nw_topo
    Comment = (none)
    Bind Queue list = {
      (none)
    }
    Job Server number list = {
      (none)

    }



Node Group: cl_ng1

    Type = cloud

    Comment = (none)

    Bind Queue list = {

      cloud_bq

    }

    Job Server number list = {

      1000

    }

    Lock State = UNLOCK

    Priority = 10

    Instances = Live: 0 / Max: 1

    Network Name = (none)

    Template = {
```

```
        (none)
    }
```

## 3.5. Job Server Startup

### 3.5.1. Startup by qmgr (1M)

It is possible to start a job server using the qmgr (1M) command from a client host, if a launcher demon of NQSV runs on the execution host.
Launcher demon can be started by following command with root user.

```
# systemctl start nqs-lchd.service
```

Launcher daemon is automatically started when execution host booted. To prevent auto start of launcher daemon, execute following command with root user.

```
# systemctl disable nqs-lchd.service
```

The "start job_server" sub-command of the qmgr (1M) command is used to start a job server. At this time, the hostname of the execution host and the job server number are designated.

```
$ qmgr -Pm
Mgr: start job_server execution_host=ehost100 job_server_id=100
Start Job_Server (ID = 100) on Execution_host (ehost100).
```

When registering execution host beforehand, it is possible to designate only the host name of the execution host or the job server number.

```
$ qmgr -Pm
Mgr: start job_server execution_host=ehost100
Start Job_Server on Execution_host (ehost100).
Mgr: start job_server job_server_id=101
Start Job_Server (ID = 101).
```

When execution host is registered with a batch server, it is not possible to designate the number besides the registered job server number for the registered execution host. In addition, it is not possible to designate the registered job server number for other execution host.

The "start job_server" sub-command with "all" parameter is used to start a job server on all execution hosts registered to the batch server.

```
$ qmgr -Pm
```

```
Mgr: start job_server all
Start Job_Server all Execution_Host.
```

To start a job server on the execution host unregistered to a batch server, the execution host name and unregistered job server number need to be specified to the "start job_server" sub-command. In this case, the execution host is registered to the batch server by the specified job server number automatically.

### 3.5.2. Startup from a Command Line

To start a job server from a command line on execution host, execute /opt/nec/nqsv/sbin/nqs_shpd by root privilege. At this time, the batch server host to be connected and the job server number need to be specified.

```
# /opt/nec/nqsv/sbin/nqs_shpd -h bsv0.example.com -n 0
```

The following command options can be specified for nqs_shpd.

| Option | Omission | Description |
|---|---|---|
| -n jsvno | Not possible | Allocate <jsvno> to job server number. |
| -h bsv_host | Not possible | Specify a batch server host name by <bsv_host>. |
| -p bsv_port | Possible | Specify a TCP port number by <bsv_port> to connect to the batch server. Default value is 602. |
| -s jsv_name | Possible | Use <jsv_name> as a job server name. Default value is JobServer*XXXX*. ("*XXXX*" is a job server No.) |

### 3.5.3. Startup by systemctl

To start job server with systemctl command, set batch server host name to BSV_HOST_NAME and job server number to JSV_NUMBER in /etc/opt/nec/nqsv/nqs_jsv.env file.

```
BSV_HOST_NAME="bsv1.example.com"
JSV_NUMBER="10"
```

After above setting, start job server with following command.

```
# systemctl start nqs-jsv.service
```

To start job server automatically when execution host is booted, execute following command with root user.

```
# systemctl enable nqs-jsv.service
```

To start the job server with same job server number and same batch server host after rebooting the execution host, edit /etc/opt/nec/nqsv/nqs_jsv.env file as following.

```
JSV_NUMBER=AUTO
```

If use this setting, no need to set BSV_HOST_NAME.

If you need to specify the additional option (the options except for -h and -n) to job server, set command line option to JSV_PARAM in /etc/opt/nec/nqsv/nqs_jsv.env file.
Following is the example to specify -s option to job server.

```
JSV_PARAM="-s MY_JSV01"
```

[Notes]
If you are using the power-saving function, set either the job server startup by a launcher daemon or the job server startup by systemd. If both are set, the job server startup may fail when the execution host is started by the power-saving function.

### 3.6. Binding Job Server and Queue

To execute job on an execution host, it is necessary to bind the job server with the queue in which requests are submitted. It is called "binding" to bind a job server with a queue, and it is called "unbinding" to unbind a job server from a queue.
Binding is performed by qmgr (1M) by root privilege. The following sub-command is used to bind a job server with a queue.

| Queue Type | Sub-command |
|---|---|
| Batch queue | bind execution_queue job_server |
| | bind execution_queue node_group |
| Interactive queue | bind interactive_queue job_server |
| | bind interactive_queue node_group |

When binding a job server with a queue, there are a way to designate the job server number directly and a way to designate a node group. The "bind xxxx_queue job_server" sub-command is used to designate the job server number.

```
$ qmgr -Pm
Mgr: bind execution_queue job_server bq1 job_server_id = 101
Bound Job_Server_ID (101) to Queue (bq1).
Mgr:
```

The "bind xxxx_queue node_group" sub-command is used to designate a node group.

```
$ qmgr -Pm
Mgr: bind execution_queue node_group bq1 node_group = ng1
Bound Node_group (ng1) to Queue (bq1).
Mgr:
```

It is possible to bind a job server with more than one queue.

When designating a node group, all job servers that belong to a designated node group are bound with a queue.

When designating a node group, a binding relation between a queue and node group is generated. Therefore, when adding execution host to a node group bound a queue, the execution host is bound with the queue automatically. In addition, when removing execution host from a node group bound a queue, the execution host is unbound from the queue automatically.

Unbinding is performed by qmgr (1M) by root privilege. The following sub-command is used to unbind a job server with a queue.

| Queue Type | Sub-command |
|---|---|
| Batch queue | unbind execution_queue job_server |
| | unbind execution_queue node_group |
| Interactive queue | unbind interactive_queue job_server |
| | unbind interactive_queue node_group |

Even if a job server bound with a queue is down, a binding relation with a queue is preserved. To unbind automatically from the queue when a job server is downed, please set automatic binding off by the "set execution_queue auto_bind_jobserver off" sub-command of qmgr (1M).

When designating a node group to unbind, a job server of all execution hosts that belong to a designated node group are unbound from a queue.

It is possible to unbind each job server that designated a node group and bound with a queue using "unbind execution_queue job_server" sub-command. A binding relation with a queue as a node group is preserved in this case.

### 3.7. Job Server Status Check

The job server status can be checked using the qstat(1) command with -S option.

```
$ qstat -S
JSVNO JobServerName   BatchServer     ExecutionHost   LINK BIND Queue    Jobs Load    Cpu
----- --------------- --------------- --------------- ---- ---- -------- ---- ----- -----
    0 JobServer0100   bsv0.example.co ehost100        UP   Y    execque1    0   0.6   0.0
    1 JobServer0101   bsv0.example.co ehost101        UP   Y    execque1    0   0.6   0.0
    2 JobServer0102   bsv0.example.co ehost102        UP   Y    execque1    0   0.6   0.0
```

Only the link-up job server is show by qstat -S option. To show all job servers status including link-down job server, set -St option.

```
$ qstat -St
JSVNO JobServerName   BatchServer     ExecutionHost   LINK BIND Queue    Jobs  Load   Cpu
----- --------------- --------------- --------------- ---- ---- -------- ---- ----- -----
    0 JobServer0100   bsv0.example.co ehost100        UP      1 execque1    0   0.6   0.0
    1 JobServer0101   bsv0.example.co ehost101        UP      1 execque1    0   0.6   0.0
    2 JobServer0102   bsv0.example.co ehost102        UP      1 execque1    0   0.6   0.0
    3 -               bsv0.example.co ehost103        DOWN    0 -           -     -     -
```

The number of the bound queue is shown to the BIND column.

Execute the qstat(1) command with -S, -f option to get the detail information of the job server.

```
$ qstat -S -f
Job Server Name: JobServer0100
    Job Server Number  = 0
    Job Server Version = R1.00 (linux)
    Batch Server   = bsv0.example.com
    Execution Host = ehost100
    LINK Batch Server = UP
    BIND Queue = BIND
    Queue = {
        execque1
    }
    Assign JobManipulator license = YES
    Network Topology Group = {
        (none)
    }
  Resource Information:
    Memory         = Assign:    1035264 Using:     925696 Maximum:    1035264
    Swap           = Assign:    1014784 Using:       3072 Maximum:    1014784
    Number of Cpus = Assign:          8 Using:          1 Maximum:          8
  Average Information:
    LOAD (Latest  1 minute ): 0.010000
    LOAD (Latest  5 minutes): 0.040000
    LOAD (Latest 15 minutes): 0.050000
    CPU  (Latest  1 minute ): 0.016000
    CPU  (Latest  5 minutes): 0.016000
    CPU  (Latest  5 minutes): 0.016000
    CPU  (Latest 15 minutes): 0.864000
  Migration_file Transfer Parameter Information:
    Interface Hostname = ehost100.example.com
    Socketbuffer Size  = (OS default)
    I/O buffer Size    = 512.0KB
```

It is possible to choose items to be displayed with -F option of qstat(1) command.

```
$ qstat -S -F jsvno,ehost,quenm
JSVNO ExecutionHost   Queue
----- --------------- --------
0     ehost100        execque1
1     ehost101        execque1
```

```
2      ehost102          execque1
```

The items that can be specified by qstat -S with -F option are as follows.

| Item | Description | Item Name |
|---|---|---|
| jsvno | Job server number | JSVNO |
| jsvnm | Job server name | JobServerName |
| bsvhost | Batch server host name | BatchServer |
| ehost | Job server execution host | ExecutionHost |
| link | Link state | LINK |
| bind | Binding state | BIND |
| quenm | Queue name | Queue |
| jobs | Number of batch jobs | Jobs |
| ldavg1 | Load average of the past for 1 minute | Load |
| cpuavg1 | CPU average of the past for 1 minute | CPU |

(About details of qstat -F option, please refer to [Operation]Customizing Information.)

It is also possible to sort the above item as a key by -O, -o option in qstat -S. (About details of a qstat -O, -o option, please refer to [Operation]Sorting Information.)

### 3.8. Job Server Stop

The job server operation is terminated by the systemd when the execution host is shut down. Send Signal SIGTERM signal to the job server to terminate during operation of the execution host.
The job server can be terminated by following command if start the job server is started by systemd (systemctl command).

```
# systemctl stop nqs-jsv.service
```

The job server can be terminated by using stop job_server sub-command of qmgr(1M). For example, please execute the following command to terminate whose job server ID (i.e. job server number) is 100.

```
$ qmgr -Pm
Mgr: stop job_server job_server_id = 100
```

[Notes]
When the job server with a running batch job is terminated, this batch job is terminated and no longer be controlled by the NQSV.
Terminate the job server while there are no running batch jobs such as after holding

requests.

### 3.9. Holding All the Requests on a Job Server

To hold the all requests running on the job server, execute the hold job_server sub-command of the qmgr(1M).
The all parent requests of the jobs running on the job server are hold executing this sub-command. Use this function when you want to clear the running job, like stopping the job server.

### 3.10. Execution Host Information

Check the execution host information, using qstat(1) command with **-E** option.

```
$ qstat -E
ExecutionHost   BatchServer     OS         Release    Hardware     VE  Load   Cpu
--------------- --------------- ---------- ---------- ---------- ---- ----- -----
ehost100        bsv0.example.co Linux      3.10.0-514 x86_64        8   0.0   0.0
ehost101        bsv0.example.co Linux      3.10.0-514 x86_64        8   0.0   0.0
ehost102        bsv0.example.co Linux      3.10.0-514 x86_64        0   0.0   0.0
```

Only the link-up job server is show by qstat -S option. To show all job servers status including link-down job server, please set -t option.
With -t option, qstat also shows state of the execution hosts (STT).

```
$ qstat -E -t
ExecutionHost   JSVNO JSV      S OS         Release    Hardware     VE  Load   Cpu STT
--------------- ----- -------- - ---------- ---------- ---------- ---- ----- ----- ---
ehost100          100 LINKUP   - Linux      3.10.0-514 x86_64        8   0.0   0.0 ACT
ehost101          101 LINKUP   - Linux      3.10.0-514 x86_64        8   0.0   0.0 ACT
ehost102          102 LINKUP   - Linux      3.10.0-514 x86_64        0   0.0   0.0 ACT
ehost103          103 LINKDOWN - --         --         --           -     -     -  INA
```

It is possible to display information by selecting and customizing each item output by qstat(1) command.
Specify items to be displayed with -F option of qstat(1) command.

```
$ qstat -E -F ehost,hwnm,ldavg1
ExecutionHost   Hardware    Load
--------------- ---------- -----
ehost100        x86_64       0.0
ehost101        x86_64       0.0
ehost102        x86_64       0.0
```

Items which can be specified by qstat -E with -F option are as follows.

| Item | Description | Item Name |
|---|---|---|

| | | |
|---|---|---|
| ehost | Execution host | ExecutionHost |
| bsvhost | Batch server host name | BatchServer |
| osnm | Name of operating system | OS |
| rel | OS release number | Release |
| hwnm | Hardware name | Hardware |
| ldavg1 | Load average of the most recent one minute | Load |
| cpuavg1 | CPU average of the most recent one minute | CPU |
| ucpu | Used CPU number | Used_CPU |
| fcpu | Free CPU number | Free_CPU |
| umem1 | Used memory size | Used_MEM1 |
| fmem1 | Free memory size | Free_MEM1 |
| uswap1 | Used swap size | Used_SWAP1 |
| fswap1 | Free swap size | Free_SWAP1 |
| quenm | Queue name | Queue |
| stt | State of the execution host | STT |

(About details of qstat -F option, please refer to [Operation]Customizing Information.)

It is also possible to sort information by any item using **-o** option or **-O** option of qstat(1) command. (About details of a qstat -O, -o option, please refer to [Operation]Sorting Information.)

To display the detail information of execution host, Use -f option with qstat -E.

```
$ qstat -E -f
Execution Host: ehost100
    Batch Server = bsv0.example.com
    Operating System = Linux (Red Hat Enterprise Linux Server release 7.3 (Maipo))
    Version =
    Release = 3.10.0-514.el7.x86_64
    Hardware = x86_64
  Vector Engine Information:
    (none)
  Resource Information:
    Memory         = Assign:    1035264 Using:     925696 Maximum:    1035264
    Swap           = Assign:    1014784 Using:       3072 Maximum:    1014784
    Number of Cpus = Assign:          8 Using:          1 Maximum:          8
  Average Information:
    LOAD (Latest  1 minute ): 0.020000
    LOAD (Latest  5 minutes): 0.050000
    LOAD (Latest 15 minutes): 0.050000
    CPU  (Latest  1 minute ): 0.016000
    CPU  (Latest  5 minutes): 0.928000
    CPU  (Latest 15 minutes): 0.928000
  Cpuset Information:
    Resource Sharing Groups = {
```

```
      (none)
    }
  Socket Resource Usage:
    NUMA Nodes = {
      (none)
    }
  Device Topology:
    (none)



Execution Host: ehost101
    Batch Server = bsv0.example.com
```

qstat -Ef displays only the execution hosts on which the job servers are UP.

To display all execution hosts registered to the batch server, please use -t option with qstat -Ef. In this case, for the execution hosts on which the job servers are down, the displayed information is limited.

And for the execution hosts on which the job servers are up, the following information is added.

- Current state of the execution host (Current State)

  Active　: The execution host is running.

  Inactive : The execution host is down.
- State transition time ( State Transition Time)
- The reason of state transition ( State Transition Reason)
- Job server number (Job Server Number )
- The link status of job server (LINK Batch Server )

  UP　　The job server is linked to the batch server.

  DOWN　The job server is not linked to the batch server.
- The information of the node management agent(Node Agent).

```
$ qstat -Etf
Execution Host: ehost100
    Batch Server = bsv0.example.com
    Current State           = Active
    State Transition Time   = Fri Dec  1 16:34:43 2017
    State Transition Reason = JSV LINKUP
    Job Server Number  = 0
    LINK Batch Server = UP
    Operating System = Linux (Red Hat Enterprise Linux Server release 7.3 (Maipo))
    Version =
    Release = 3.10.0-514.el7.x86_64
    Hardware = x86_64
    Node Agent = (none)
    Substitute Status = Normal
  Vector Engine Information:
    (none)
```

```
  Resource Information:
    Memory         = Assign:    1035264 Using:     925696 Maximum:    1035264
    Swap           = Assign:    1014784 Using:       3072 Maximum:    1014784
    Number of Cpus = Assign:          8 Using:          1 Maximum:          8
  Average Information:
    LOAD (Latest  1 minute ): 0.020000
    LOAD (Latest  5 minutes): 0.050000
    LOAD (Latest 15 minutes): 0.050000
    CPU  (Latest  1 minute ): 0.016000
    CPU  (Latest  5 minutes): 0.928000
    CPU  (Latest 15 minutes): 0.928000
  Cpuset Information:
    Resource Sharing Groups = {
      (none)
    }
  Socket Resource Usage:
    NUMA Nodes = {
      (none)
    }
  Device Topology:
    (none)

Execution Host: ehost101
    Batch Server = bsv0.example.com
        :
        :

Execution Host: ehost103
    Batch Server = bsv0.example.com
    Current State           = Inactive
    State Transition Time   = Mon Mar 3 15:00:00 2017
    State Transition Reason = JSV LINKDOWN
    Job Server Number  = 103
    LINK Batch Server = DOWN
```

### 3.11. VE node Information

This function is available only for the environment whose execution host is SX-Aurora TSUBASA system.

Using qstat --venode shows the information about the VE nodes.

By specifying the VI host name as the parameter for this option, only the information of VEs incorporated on the specified VI (VH) can be displayed.

If no parameter is specified, information about the VEs on all VIs (VHs) the JSV is linking up are displayed.

```
$ qstat --venode
```

```
VectorIsland VE_No  Cores Memory Status        OS_Status
------------ ----- ------ ------ ------------- ----------
ehost100         0     10   48GB ONLINE        ONLINE
ehost100         1     10   48GB UNINITIALIZED OFFLINE
ehost100         2     10   48GB OFFLINE       OFFLINE
ehost100         3     10   48GB MAINTENANCE   OFFLINE
ehost101         0     10   48GB UNAVAILABLE   OFFLINE
ehost101         1     10   48GB ONLINE        OFFLINE
ehost101         2     10   48GB ONLINE        ONLINE
ehost101         3     10   48GB ONLINE        ONLINE
```

# 4. Queue Management

## 4.1. Batch Queue

### 4.1.1. Create Batch Queue

Batch queue is the queue to control batch requests execution.

Create a new batch queue using the "create execution_queue" sub-command of qmgr(1M) command.

```
$ qmgr -Pm
Mgr: create execution_queue = exec1 priority = 20
```

Batch queue "exec1" will be created with above procedure. The queue priority should be specified to "priority" above.

### 4.1.2. Batch Queue Configuration

(1) Resource Limit

It is possible to limit resource consumption of request in a batch queue. Batch requests submitted will be rejected by the batch queue if resource consumption specified to the batch request exceeds resource limit set to the batch queue.

Using resource limit, it is possible to classify queues into such as queues that permit large consumption of resources and into queues that allow only limited consumption.

Set resource limit attributes using set execution_queue sub-command of qmgr(1M) command. Resource limits and corresponding set sub-commands of the qmgr command are shown as follows, including whether they are valid on each execution host. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| For Each Process | |
| CPU time limit | set execution_queue per_prc cpu_time_limit |
| Open file number limit(*1) | set execution_queue per_prc open_file_number_limit |
| Memory size limit | set execution_queue per_prc memory_size_limit |
| Data size limit | set execution_queue per_prc data_size_limit |
| Stack size limit | set execution_queue per_prc stack_size_limit |
| Core file size limit | set execution_queue per_prc core_size_limit |
| File size limit | set execution_queue per_prc file_size_limit |
| Virtual memory size limit | set execution_queue per_prc virtual_memory_size_limit |
| VE CPU time limit | set execution_queue per_prc vecpu_time_limit |
| VE memory size limit | set execution_queue per_prc vememory_size_limit |
| For Each Job | |
| CPU time limit | set execution_queue per_job cpu_time_limit |
| CPU resident number limit | set execution_queue per_job cpu_number_limit |
| Memory size limit | set execution_queue per_job memory_size_limit |
| Virtual memory size limit | set execution_queue per_job virtual_memory_size_limit |
| Number of GPU Limit | set execution_queue per_job gpu_number_limit |
| For Each Request | |

| | |
|---|---|
| Elapse time limit | set execution_queue per_req elapse_time_limit |

(*1) When "unlimited" is specified, "1024" is applied to the upper limit of an execution host according to the specification restriction of Linux OS.

It is also possible to set elapse time limit per individually designated group/user name, please refer to 11. Limit per Group and User.

An example of changing a file size limit for each process is shown below. This operation sets the file size limitation of each process of the exec1 queue to 100.5 KB.

```
$ qmgr -Po
Mgr(bsv1.example.com): set execution_queue per_prc file_size_limit = (100.5kb)  exec1
Set Permanent File Size (Per-Process) limit: bq1
```

Resource limits set on queues can be confirmed by qstat(1) command -Q, -f option (Resources Limits). (Please refer to 4.5.1. Batch queue)

Memory size resource limits can also be controlled using cgroups. (See 2.3.19 Memory Management with memory cgroup)

**Resource Limitation per VE Node**

For a request to be inputted to a batch queue, the resource usage limit can be set to each VE node to be assigned to a job.

For the VE node resource limitation can be set the upper limit and the lower limit, making it possible to limit the usage according to a range. If the resource usage specified when inputting a request is out of the value range of the resource limitation that is set to a queue, the request input to the queue is rejected.

Set resource limit attributes using set execution_queue venode sub-command of qmgr(1M) command. Resource limits and corresponding set sub-commands of the qmgr command are shown as follows, including whether they are valid on each execution host. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| VE CPU time limit | set execution_queue venode vecpu_time_range = (<min>,<Max>[,<warn>]) <queue-name> |
| VE memory size limit | set execution_queue venode vememory_size_range = (<min>,<max>[,<warn>]) <queue-name> |

For <min>, specify the minimum value. For <max>, specify the maximum value. Both must be an integer. For <min>, specify a value smaller than <max>. If the same value is specified for both <min> and <max>, only that value can be specified when inputting a request.

Resource limits set on queues can be confirmed by qstat(1) command -Q, -f option (VE Node Resource Ranges).

**Resource Limitation per Logical Host**

By using a specification method different from the above, for a request to be inputted to a batch queue, the resource usage limit can be set to each logical host to be assigned to a job.

For the logical host resource limitation, not only the upper limit but also the lower limit can be set in contrast to the conventional resource limitation per job, making it possible to limit the usage according to a range. If the resource usage specified when inputting a request is out of the value range of the resource limitation that is set to a queue, the request input to the queue is rejected.

The operations related to the resource limitation set to a logical host are the same as the conventional operations related to the resource limitation set to a job.

Set resource limit attributes using set execution_queue lhost sub-command of qmgr(1M) command. Resource limits and corresponding set sub-commands of the qmgr command are shown as follows, including whether they are valid on each execution host. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| Number of VE node Limit | set execution_queue lhost ve_number_range = (<min>,<max>) <queue-name> |
| CPU number limit | set execution_queue lhost cpu_number_range = (<min>,<max>) <queue-name> |
| Number of GPU Limit | set execution_queue lhost gpu_number_range = (<min>,<max>) <queue-name> |
| CPU time limit | set execution_queue lhost cpu_time_range = (<min>,<max>[,<warn>]) <queue-name> |
| Memory size limit | set execution_queue lhost memory_size_range = (<min>,<max>[,<warn>]) <queue-name> |
| Virtual memory size limit | set execution_queue lhost virtual_memory_size_range = (<min>,<max>[,<warn>]) <queue-name> |
| VE CPU time limit | set execution_queue lhost vecpu_time_range = (<min>,<Max>[,<warn>]) <queue-name> |
| VE memory size limit | set execution_queue lhost vememory_size_range = (<min>,<max>[,<warn>]) <queue-name> |
| Stdout size limit | set execution_queue lhost stdout_size_range = (<min>,<max> [,<warn>]) <queue-name> |
| Stderr size limit | set execution_queue lhost stderr_size_range = (<min>,<max> [,<warn>]) <queue-name> |

For <min>, specify the minimum value. For <max>, specify the maximum value. Both must be an integer. For <min>, specify a value smaller than <max>. If the same value is specified for

both <min> and <max>, only that value can be specified when inputting a request.

Resource limits set on queues can be confirmed by qstat(1) command -Q, -f option (Logical Host Resource Ranges).

These resource limitations of a logical host (excluding the limitation on the number of VE nodes, the limitation on the time of VE CPU and the limitation on the size of VE memory) can also be set by using the following methods (by specifying per_job) to set the resource limitations of a job.

```
set execution_queue per_job cpu_number_limit = <max> <queue-name>
set execution_queue per_job gpu_number_limit = <max> <queue-name>
set execution_queue per_job cpu_time_limit = (<max>,[<warn>]) <queue-name>
set execution_queue per_job memory_size_limit = (<max>,[<warn>]) <queue-name>
set execution_queue per_job virtual_memory_size_limit = (<max>,[<warn>]) <queue-name>
```

When the resource limitations are set by using the above methods, it is assumed that 1 for cpu_number_limit, and the others are 0 (zero) is specified for <min> of all resource limitations of a logical host. For <max> and <warn>, the specified values are used.

For both specification methods using lhost and per_job, the value you specify later becomes effective.

**HCA port number limit**

It is possible to limit the HCA port number for job of request in a batch queue. Batch requests submitting will be rejected by the batch queue if HCA port number specified to the batch request exceeds this limit set to the batch queue.
HCA port number limit for batch queue is specified by "set execution_queue hca_number_range" sub-command of qmgr(1M). The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| HCA port number limit | set execution_queue hca_number_range = (<min>,<max>) mode=<hca-mode> <queue-name> |

For <min>, specify the minimum value. For <max>, specify the maximum value. Both must be an integer. For <min>, specify a value smaller than <max>. If the same value is specified for both <min> and <max>, only that value can be specified when submitting a request.

(2) Resource Default

Resource default is an attribute used as a resource limit value for a request where a resource limit value is not explicitly specified when submitted to a queue. This attribute will not be referred when a resource limit value is set on requests.

Set the resource default attribute using set sub-command of qmgr(1M) command. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| For Each Process | |
| CPU time limit | set execution_queue standard per_prc cpu_time_limit |
| Open file number limit | set execution_queue standard per_prc open_file_number_limit |
| Memory size limit | set execution_queue standard per_prc memory_size_limit |
| Data size limit | set execution_queue standard per_prc data_size_limit |
| Stack size limit | set execution_queue standard per_prc stack_size_limit |
| Core file size limit | set execution_queue standard per_prc core_size_limit |
| File size limit | set execution_queue standard per_prc file_size_limit |
| Virtual memory size limit | set execution_queue standard per_prc virtual_memory_size_limit |
| VE CPU time limit | set execution_queue standard per_prc vecpu_time_limit |
| VE memory size | set execution_queue standard per_prc vememory_size_limit |
| For Each Job | |
| CPU time limit | set execution_queue standard per_job cpu_time_limit |
| CPU number limit | set execution_queue standard per_job cpu_number_limit |
| Memory size limit | set execution_queue standard per_job memory_size_limit |
| Virtual memory size limit | set execution_queue standard per_job virtual_memory_size_limit |
| Number of GPU Limit | set execution_queue standard per_job gpu_number_limit |
| For Each Request | |
| Elapse time limit | set execution_queue standard per_req elapse_time_limit |

An example of setting a resource default at a permanent file size for each process is shown below. The file-size resource default for each process of the queue "exec1" is 100.5Kbytes. The operator privilege is necessary to set the default.

```
$ qmgr -Po
Mgr: set execution_queue standard per_prc file_size_limit = (100.5kb)  exec1
Set standard Permanent File Size (Per-Process): glbque1
```

**Resource Default per Logical Host**

By using a specification method different from the above, the resource default of a logical host to be assigned to a job can be set as an attribute value of a queue. The operations related to the resource default set to a logical host when inputting a request are the same as those related to the resource default set to a job.

Resource limits and corresponding set sub-commands of the qmgr command are shown as

follows. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| Number of VE node Limit | set execution_queue standard lhost ve_number_limit = <num> <queue-name> |
| CPU number limit | set execution_queue standard lhost cpu_number_limit = <num> <queue-name> |
| Number of GPU Limit | set execution_queue standard lhost gpu_number_limit = <num> <queue-name> |
| CPU time limit | set execution_queue standard lhost cpu_time_limit = (<time>) <queue-name> |
| Memory size limit | set execution_queue standard lhost memory_size_limit = (<size>) <queue-name> |
| Virtual memory size limit | set execution_queue standard lhost virtual_memory_size_limit = (<size>) <queue-name> |
| VE CPU time limit | set execution_queue standard lhost vecpu_time_limit = (<time>) <queue-name> |
| VE memory size limit | set execution_queue standard lhost vememory_size_limit = (<size>) <queue-name> |
| Stdout size limit | set execution_queue standard lhost stdout _size_limit = (<size>) <queue-name> |
| Stderr size limit | set execution_queue standard lhost stderr _size_limit = (<size>) <queue-name> |

The resource default must be between the lower limit (min) and upper limit (max) of the resource limitation. For the items for which a unit must be added (cpu_time, memory_size, virtual_memory_size, vecpu_time, vememory_size, stdout_size, stderr_size), the specified value must be enclosed in ().

These resource defaults of a logical host (excluding the limitation on the number of VE nodes, the limitation on the time of VE CPU and the size of VE memory) can also be set by using the following conventional methods (by specifying per_job) to set the resource limitations of a job.

```
set execution_queue standard per_job cpu_number_limit = <num> <queue-name>
set execution_queue standard per_job gpu_number_limit = <num> <queue-name>
set execution_queue standard per_job cpu_time_limit = (<time>) <queue-name>
set execution_queue standard per_job memory_size_limit = (<size>) <queue-name>
set execution_queue standard per_job virtual_memory_size_limit = (<size>) <queue-name>
```

**Resource Default per VE Node**

The resource default of a VE node to be assigned to a job can be set as an attribute value of a queue. The operations related to the resource default set to a VE node when inputting a request are the same as those related to the resource default set to a job.

Resource limits and corresponding set sub-commands of the qmgr command are shown as follows. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| VE CPU time limit | set execution_queue standard venode vecpu_time_limit = (<time>) <queue-name> |
| VE memory size limit | set execution_queue standard venode vememory_size_limit = (<size>) <queue-name> |

The resource default must be between the lower limit (min) and upper limit (max) of the resource limitation. For the items for which a unit must be added (cpu_time, memory_size, virtual_memory_size), the specified value must be enclosed in ().

**HCA port number default**

HCA port number default is an attribute used as a HCA port number limit value for a request where a HCA port number limit value is not explicitly specified when submitted to a queue. This attribute will not be referred when a HCA port number limit value is set on requests.

Set the HCA port number default attribute using "set execution_queue standard hca_number" sub-command of qmgr(1M) command. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| HCA port number default | set execution_queue standard hca_number = <num> mode=<hca-mode> <queue-name> |

The default value must be bigger than <min> value and smaller than <max> value.

(3) Queue Priority

Queue priority is a priority among queues and is used when selecting which request should be executed first.

The queue priority of the batch queue is specified at the time of creating batch queue. It is also possible to change it by the "set execution_queue priority" sub-command of qmgr.

```
$ qmgr -Po
Mgr: set execution_queue priority = 10 exec1
Set Priority: exec1
```

The queue priority of the batch queue "exec1" is changed to 10. The operator privilege is necessary to change queue priority.

(4) Kernel Parameter

The kernel parameters that are set to the batch queues are inherited by the batch jobs, and they will be the parameters for scheduling the jobs by kernel.

Set the kernel parameters using set sub-command of qmgr(1M) command. The operator privilege is necessary. The following table shows the kernel parameters, including corresponding set sub-command of the qmgr command, and their default values.

| Kernel parameter | qmgr(1M) sub-command | Default |
|---|---|---|
| Nice Value | set execution_queue kernel_param nice | 0 |
| RSG number(*1) | set execution_queue kernel_param rsg_number | 0 |

(*1) It is enabled when the socket scheduling and CPUSET feature are enabled.

The following sample is available for setting a nice value. The nice value of the batch queue "exec1" is changed to "5".

```
$ qmgr -Po
Mgr: set execution_queue kernel_param nice = 5 exec1
Set Nice Value (Kernel-Parameter): exec1
```

(5) Submit Limit

It is possible to set a limit to the number of requests that can be submitted to a batch queue. The number of requests that can be submitted means the total number of requests existent in a queue at the same time. With this submit limit per batch queue, more detailed submit limit of requests can be configured than submit limit per batch system.

(For the details of submit limit per batch system, please refer to 2.3.3. Submit Limit.)

The submit limit per batch queue can also be specified per whole queue, group and user. Set the limit to the number of requests using sub-commands of qmgr(1M). The operator privilege or higher is necessary to set this limit.

The sub-commands and the default values are as follows.

| Attribute | qmgr (1M) sub-command | Default |
|---|---|---|
| The number of requests that can be submitted to a queue | set execution_queue submit_limit | 0 (unlimited) |
| The number of requests that one group can submit to a queue | set execution_queue group_submit_limit | 0 (unlimited) |
| The number of requests that one user can submit to a queue | set execution_queue user_submit_limit | 0 (unlimited) |

It is also possible to set submit number limit per individually designated group/user name, please refer to 11. Limit per Group and User.

Submit limits set on queues can be confirmed by qstat(1) command -Q, -f option as follows.(Please refer to 4.5.1. Batch queue.)

- The number of requests that can be submitted to a queue: Submit Number Limit
- The number of requests that one user can submit to a queue: Submit User Number Limit
- The number of requests that one group can submit to a queue: Submit Group Number Limit

(For each item, limit values set are shown or if no limits are set "UNLIMITED" is shown.)

If both submit limit per batch system and submit limit per queue are set, requests cannot be submitted at the point of exceeding either limit.

(6) Hold Privilege

Hold privilege is the privilege that can hold the requests. (For the details of holding batch requests, please refer to [Operation]Batch Request Hold.)

By setting a hold privilege to the batch queue, only a hold request issued by user with higher privilege than the privilege that is set for the queue can be accepted.

Set the hold privilege per batch queue using the "set execution_queue hold_privilege" sub-command of the qmgr(1M) command.

An example of setting the hold privilege as higher than operator privilege is shown below:

```
$ qmgr -Po
Mgr: set execution_queue hold_privilege = operator exec1
Set Hold Privilege.queue: exec1
```

To delete the hold privilege, execute the "delete execution_queue hold_privilege" sub-command of the qmgr(1M) command.

(7) Suspend Privilege

Suspend privilege is the privilege that can suspend requests. (For the details of suspending requests, please refer to [Operation]Batch Request Suspend.) By setting a suspend privilege to the batch queue, only a suspend request issued by user with higher privilege than the privilege that is set for the queue can be accepted. Set the suspend privilege per batch queue using the "set execution_queue suspend_privilege" sub-command of the qmgr(1M) command.

An example of setting the suspend privilege as higher than operator privilege is shown below:

```
$ qmgr -Po
Mgr: set execution_queue suspend_privilege = operator exec1
Set Suspend Privilege.queue: exec1
```

To delete the suspend privilege, execute the "delete execution_queue suspend_privilege" sub-command of the qmgr(1M) command.

(8) User EXIT Script

User Exit script file is a shell script file executed when the request transits through a specific

state. It is executed when using the User Exit Function. (Please refer to 5.1.2. User EXIT.)
Set the user exit file per batch queue using the "set execution_queue userexit" sub-command of the qmgr(1M) command.
An example of setting the script file script1 that is to be executed at pre-running state for requests of the batch queue "exec1" is shown below.

```
$ qmgr -Po
Mgr: set execution_queue userexit location = pre-running script = (script1) queue =
exec1
Set Userexit Script: exec1
```

Execute the "delete execution_queue userexit" sub-command of the qmgr (1M) command to delete the user exit script setting.
An example of deleting the specified script of the batch queue "exec1" to run at pre-running is shown below.

```
$ qmgr -Po
Mgr: delete execution_queue userexit location = pre-running queue = exec1
Delete Userexit Script: exec1
```

(9) Setting Time-out of the User EXIT Execution

When set time-out time of the User EXIT execution, the User EXIT occurred any trouble like a stall, it is possible to send a KILL signal to the User EXIT and abort the execution. Time-out time is set by the sub-command of qmgr (1M).
An example of setting time-out time of the User EXIT execution to 900 seconds to batch queue "exec1" is shown below.

```
$ qmgr -Po
Mgr: set execution_queue userexit_timeout = 900 exec1
Set UserExit Timeout: exec1
```

A default value of time-out time of the User EXIT execution is 0 (off). In case of 0, time-out is not occurred.

(10) Limitation of the Job Number

It is possible to limit the number of jobs created from request. Set limitation of the number of jobs per batch queue using the "set execution_queue jobs_range" sub-command of the qmgr(1M) command.
An example of setting a range of 100 to 500 for the number of jobs in the batch queue "exec1" is shown below.

```
$ qmgr -Po
Mgr: set execution_queue jobs_range = (100,500) exec1
Set Jobs Range: exec1
```

It is also possible to set limitation of the job number per individually designated group/user name, please refer to 11. Limit per Group and User.

(11) Limitation of Request Priority Range

Batch queues can be set the range of request priority permitted to submit requests for each user privilege. Set limitation of the request priority range per batch queue using the "set execution_queue request_priority_range" sub-command of the qmgr(1M) command.

An example of setting a range of -100 to 0 for request priorities for the User privilege of the execution queue "exec1" is shown below.

```
$ qmgr -Po
Mgr: set execution_queue request_priority_range user = (-100,0) exec1
Set Request Priority Range (User): exec1
```

(12) Limitation of the Subrequest Number

It is possible to limit the number of subrequests created from parametric request. Set limitation of the number of subrequests per batch queue using the "set execution_queue subrequest_limit" sub-command of the qmgr(1M) command. An example of setting 100 for the number of subrequests in the batch queue "exec1" is shown below.

```
$ qmgr -Po
Mgr: set execution_queue subrequest_limit = 100 exec1
Set queue subrequest limit: exec1
```

The default number of the subrequests is 1000.

(13) Allowance for exclusive execution request

It is able to configure the allowance for exclusive execution request (which is submitted by qsub --exclusive) to the queue. The exclusive execution request always executed as 1 logical host(job) per 1 host and it ignore the resource limit (CPU, MEMORY, Virtual MEMORY, GPU, VE) for logical host of the queue. Therefore this configuration can limit the submission of that request.

Allowance for exclusive execution request for batch queue can be set by following qmgr sub-command with operator privilege.

```
set execution_queue exclusive_submit {on | off} <queue>
```

off rejects the submission of the request with --exclusive option. on allows the submission of the request with --exclusive option. The default is off.

(14) Disabling the stage-out

It is able to configure the setting for disabling stage-out to the queue. If this option is set to off, the stage-out of the request which submitted to the queue is not executed, and it increase the throughput of request processing.

This option can be set by following qmgr sub-command with operator privilege.

```
set execution_queue file_stageout = { on | off }   <queue>
```

"off" do not execute stage-out for the request even if the stage-out file is specified to the request.

The default is on.

Notes

If this option is set to "off", the behavior of NQSV changes as following to prioritize the batch server processing speed.

- The standard out / error file is not stage-out to the host which the request submitted. Please use qsub –e option or –o option to specify the output path. This path is treated as the path on the execution host.
- Check the directory for the standard out / error file output is exist on the execution host. If the directory is not exist, the execution of the job fails.
- The meta character %[0n]j and %f cannot be used for the file name on the qsub –e option and –o option.
- The request log is not output
- To executing multi-node NECMPI request, use qsub –f option to output by each rank.

## 4.2. Interactive Queue

### 4.2.1. Create Interactive Queue

Interactive queue is the queue to control execution of interactive request.

Create a new interactive queue using the "create interactive_queue" sub-command of qmgr(1M) command.

```
$ qmgr -Pm
Mgr: create interactive_queue = inter1 priority = 20
```

Batch queue "inter1" will be created with above procedure. The queue priority should be specified to "priority" above.

### 4.2.2. Interactive Queue Configuration

(1) Resource Limit

It is possible to limit resource consumption of request in an interactive queue. Resource limits and corresponding set sub-commands of the qmgr command are shown as follows, including whether they are valid on each execution host.

| Resource | qmgr(1M) sub-command |
|---|---|
| For Each Process | |
| CPU time limit | set interactive_queue per_prc cpu_time_limit |
| Open file number limit (*1) | set interactive_queue per_prc open_file_number_limit |
| Memory size limit | set interactive_queue per_prc memory_size_limit |
| Data size limit | set interactive_queue per_prc data_size_limit |
| Stack size limit | set interactive_queue per_prc stack_size_limit |
| Core file size limit | set interactive_queue per_prc core_size_limit |
| File size limit | set interactive_queue per_prc file_size_limit |

| | |
|---|---|
| Virtual memory size limit | set interactive_queue per_prc virtual_memory_size_limit |
| VE CPU time limit | set interactive_queue per_prc vecpu_time_limit |
| VE memory size limit | set interactive_queue per_prc vememory_size_limit |
| For Each Job | |
| CPU time limit | set interactive_queue per_job cpu_time_limit |
| CPU number limit | set interactive_queue per_job cpu_number_limit |
| Memory size limit | set interactive_queue per_job memory_size_limit |
| Virtual memory size limit | set interactive_queue per_job virtual_memory_size_limit |
| Number of GPU Limit | set interactive_queue per_job gpu_number_limit |
| For Each Request | |
| Elapse time limit | set interactive_queue per_req elapse_time_limit |

(*1) 1024 is applied when it set to unlimited.

It is also possible to set elapse time limit per individually designated group/user name, please refer to 11. Limit per Group and User.

Memory size resource limits can also be controlled using cgroups. (See 2.3.19 Memory Management with memory cgroup)

**Resource Limitation per VE Node**

For a request to be inputted to an interactive queue, the resource usage limit can be set to each VE node to be assigned to a job. For the VE node resource limitation can be set the upper limit and the lower limit, making it possible to limit the usage according to a range. If the resource usage specified when inputting a request is out of the value range of the resource limitation that is set to a queue, the request input to the queue is rejected. The operations related to the resource limitation set to a VE node are the same as the conventional operations related to the resource limitation set to a job.

Set resource limit attributes using set interactive_queue venode sub-command of qmgr(1M) command. Resource limits and corresponding set sub-commands of the qmgr command are shown as follows, including whether they are valid on each VE node. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| VE CPU time limit | set interactive_queue venode vecpu_time_range = (<min>,<max>[,<warn>] <queue-name> |
| VE memory size limit | set interactive_queue venode vememory_size_range = (<min>,<max>[,<warn>] <queue-name> |

For <min>, specify the minimum value. For <max>, specify the maximum value. Both must be an integer. For <min>, specify a value smaller than <max>. If the same value is specified for both <min> and <max>, only that value can be specified when inputting a request.

Resource limits set on queues can be confirmed by qstat(1) command -Q, -f option (VE Node Resource Ranges).

**Resource Limitation per Logical Host**

By using a specification method different from the above, for a request to be inputted to an interactive queue, the resource usage limit can be set to each logical host to be assigned to a job.

For the logical host resource limitation, not only the upper limit but also the lower limit can be set in contrast to the conventional resource limitation per job, making it possible to limit the usage according to a range. If the resource usage specified when inputting a request is out of the value range of the resource limitation that is set to a queue, the request input to the queue is rejected.

The operations related to the resource limitation set to a logical host are the same as the conventional operations related to the resource limitation set to a job.

Set resource limit attributes using set interactive_queue lhost sub-command of qmgr(1M) command. Resource limits and corresponding set sub-commands of the qmgr command are shown as follows, including whether they are valid on each execution host. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| Number of VE node Limit | set interactive_queue lhost ve_number_range = (<min>,<max>) <queue-name> |
| CPU number limit | set interactive_queue lhost cpu_number_range = (<min>,<max>) <queue-name> |
| Number of GPU Limit | set interactive_queue lhost gpu_number_range = (<min>,<max>) <queue-name> |
| CPU time limit | set interactive_queue lhost cpu_time_range = (<min>,<max>[,<warn>]) <queue-name> |
| Memory size limit | set interactive_queue lhost memory_size_range = (<min>,<max>[,<warn>]) <queue-name> |
| Virtual memory size limit | set interactive_queue lhost virtual_memory_size_range = (<min>,<max>[,<warn>]) <queue-name> |
| VE CPU time limit | set interactive_queue lhost vecpu_time_range = (<min>,<max>[,<warn>] <queue-name> |
| VE memory size limit | set interactive_queue lhost vememory_size_range = (<min>,<max>[,<warn>] <queue-name> |

For <min>, specify the minimum value. For <max>, specify the maximum value. Both must be an integer. For <min>, specify a value smaller than <max>. If the same value is specified for both <min> and <max>, only that value can be specified when inputting a request.

Resource limits set on queues can be confirmed by qstat(1) command -Q, -f option (Logical Host Resource Ranges).

These resource limitations of a logical host (excluding the limitation on the number of VE nodes, the limitation on the time of VE CPU and the limitation on the size of VE memory) can also be set by using the following methods (by specifying per_job) to set the resource limitations of a job.

```
set interactive_queue per_job cpu_number_limit = <max> <queue-name>
set interactive_queue per_job gpu_number_limit = <max> <queue-name>
set interactive_queue per_job cpu_time_limit = (<max>,[<warn>]) <queue-name>
set interactive_queue per_job memory_size_limit = (<max>,[<warn>]) <queue-name>
set interactive_queue per_job virtual_memory_size_limit = (<max>,[<warn>]) <queue-name>
```

When the resource limitations are set by using the above methods, it is assumed that 0 (zero) is specified for <min> of all resource limitations of a logical host. For <max> and <warn>, the specified values are used.

For both specification methods using lhost and per_job, the value you specify later becomes effective.

**HCA port number limit**

It is possible to limit the HCA port number for job of request in an interactive queue.
Interactive requests submitting will be rejected by the interactive queue if HCA port number specified to the interactive request exceeds this limit set to the interactive queue.
HCA port number limit for interactive queue is specified by "set interactive_queue hca_number_range" sub-command of qmgr(1M). The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| HCA port number limit | set interactive_queue hca_number_range = (<min>,<max>) mode=<hca-mode> <queue-name> |

For <min>, specify the minimum value. For <max>, specify the maximum value. Both must be an integer. For <min>, specify a value smaller than <max>. If the same value is specified for both <min> and <max>, only that value can be specified when submitting a request.

(2) Resource Default

Resource default is an attribute used as a resource limit value for a request where a resource limit value is not explicitly specified when submitted to a queue.
Set the resource default attribute using set sub-command of qmgr(1M) command.

| Resource | qmgr(1M) sub-command |
|---|---|

| For Each Process | |
|---|---|
| CPU time limit | set interactive_queue standard per_prc cpu_time_limit |
| Open file number limit | set interactive_queue standard per_prc open_file_number_limit |
| Memory size limit | set interactive_queue standard per_prc memory_size_limit |
| Data size limit | set interactive_queue standard per_prc data_size_limit |
| Stack size limit | set interactive_queue standard per_prc stack_size_limit |
| Core file size limit | set interactive_queue standard per_prc core_size_limit |
| File size limit | set interactive_queue standard per_prc file_size_limit |
| Virtual memory size limit | set interactive_queue standard per_prc virtual_memory_size_limit |
| VE CPU time limit | set interactive_queue standard per_prc vecpu_time_limit |
| VE memory size limit | set interactive_queue standard per_prc vememory_size_prc |
| For Each Job | |
| CPU time limit | set interactive_queue standard per_job cpu_time_limit |
| CPU number limit | set interactive_queue standard per_job cpu_number_limit |
| Memory size limit | set interactive_queue standard per_job memory_size_limit |
| File capacity limit | set interactive_queue standard per_job file_capacity_limit |
| Virtual memory size limit | set interactive_queue standard per_job virtual_memory_size_limit |
| Number of GPU Limit | set interactive_queue standard per_job gpu_number_limit |
| For Each Request | |
| Elapse time limit | set interactive_queue standard per_req elapse_time_limit |

**Resource Default per Logical Host**

By using a specification method different from the above, the resource default of a logical host to be assigned to a job can be set as an attribute value of a queue. The operations related to the resource default set to a logical host when inputting a request are the same as those related to the resource default set to a job.

Resource limits and corresponding set sub-commands of the qmgr command are shown as follows. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| Number of VE node Limit | set interactive_queue standard lhost ve_number_limit = <num> <queue-name> |
| CPU number limit | set interactive_queue standard lhost cpu_number_limit = <num> <queue-name> |
| Number of GPU Limit | set interactive_queue standard lhost gpu_number_limit = <num> <queue-name> |
| CPU time limit | set interactive_queue standard lhost cpu_time_limit = (<time>) <queue-name> |
| Memory size limit | set interactive_queue standard lhost memory_size_limit = (<size>) <queue-name> |
| Virtual memory size limit | set interactive_queue standard lhost virtual_memory_size_limit = (<size>) <queue-name> |
| VE CPU time limit | set interactive_queue standard lhost vecpu_time_limit = (<time>) <queue-name> |

| | |
|---|---|
| VE memory size limit | set interactive_queue standard lhost vememory_size_limit = (<size>) <queue-name> |

The resource default must be between the lower limit (min) and upper limit (max) of the resource limitation. For the items for which a unit must be added (cpu_time, memory_size, virtual_memory_size), the specified value must be enclosed in ().

These resource defaults of a logical host (excluding the limitation on the number of VE nodes, the limitation on the time of VE CPU and the limitation on the size of VE memory) can also be set by using the following conventional methods (by specifying per_job) to set the resource limitations of a job.

```
set interactive_queue standard per_job cpu_number_limit = <num> <queue-name>
set interactive_queue standard per_job gpu_number_limit = <num> <queue-name>
set interactive_queue standard per_job cpu_time_limit = (<time>) <queue-name>
set interactive_queue standard per_job memory_size_limit = (<size>) <queue-name>
set interactive_queue standard per_job virtual_memory_size_limit = (<size>) <queue-name>
```

**Resource Default per VE Node**

The resource default of a VE node to be assigned to a job can be set as an attribute value of a queue. The operations related to the resource default set to a VE node when inputting a request are the same as those related to the resource default set to a job.

Resource limits and corresponding set sub-commands of the qmgr command are shown as follows. The operator privilege is necessary to set the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| VE CPU time limit | set interactive_queue standard venode vecpu_time_limit = (<time>) <queue-name> |
| VE memory size limit | set interactive_queue standard venode vememory_size_limit = (<size>) <queue-name> |

The resource default must be between the lower limit (min) and upper limit (max) of the resource limitation. For the items for which a unit must be added (cpu_time, memory_size, virtual_memory_size), the specified value must be enclosed in ().

**HCA port number default**

HCA port number default is an attribute used as a HCA port number limit value for a request where a HCA port number limit value is not explicitly specified when submitted to a queue. This attribute will not be referred when a HCA port number limit value is set on requests.

Set the HCA port number default attribute using "set interactive_queue standard hca_number" sub-command of qmgr(1M) command. The operator privilege is necessary to set

the limit.

| Resource | qmgr(1M) sub-command |
|---|---|
| HCA port number default | set interactive_queue standard hca_number = <num> mode=<hca-mode> <queue-name> |

The default value must be bigger than <min> value and smaller than <max> value.

(3) Queue Priority

Queue Priority is a priority among queues, and it is possible to put the order of priority of scheduling with other queues.

The queue priority of the interactive queue is specified at the time of creating interactive queue. It is also possible to change it by the "set interactive_queue priority" sub-command of qmgr.

```
$ qmgr -Po
Mgr: set interactive_queue priority = 10 inter1
Set Priority: inter1
```

The queue priority of the interactive queue "inter1" is changed to 10. The operator privilege is necessary to change queue priority.

(4) Kernel Parameter

Set the kernel parameters using set sub-command of qmgr(1M) command. The operator privilege is necessary.

The following table shows the kernel parameters, including corresponding set sub-command of the qmgr command, and their default values.

| Kernel parameter | qmgr(1M) sub-command | Default |
|---|---|---|
| Nice Value | set interactive_queue kernel_param nice | 0 |
| RSG number(*1) | set interactive_queue kernel_param rsg_number | 0 |

(*1) It is enabled when the socket scheduling and CPUSET feature are enabled.

The following sample is available for setting a nice value. The nice value of the interactive queue "inter1" is changed to "5".

```
$ qmgr -Po
Mgr: set interactive_queue kernel_param nice = 5 inter1
Set Nice Value (Kernel-Parameter): inter1
```

(5) Submit Limit

It is possible to set a limit to the number of requests that can be submitted to an interactive queue. The number of requests that can be submitted means the total number of requests

existent in a queue at the same time. With this submit limit per interactive queue, more detailed submit limit of requests can be configured than submit limit per batch system.
(For the details of submit limit per batch system, please refer to 2.3.3. Submit Limit.)
The submit limit per interactive queue can also be specified per whole queue, group and user. Set the limit to the number of requests using sub-commands of qmgr(1M). The operator privilege or higher is necessary to set this limit.
The sub-commands and the default values are as follows.

| Attribute | qmgr(1M) sub-command | Default |
|---|---|---|
| The number of requests that can be submitted to a queue | set interactive_queue submit_limit | 0 (unlimited) |
| The number of requests that one group can submit to a queue | set interactive_queue group_submit_limit | 0 (unlimited) |
| The number of requests that one user can submit to a queue | set interactive_queue user_submit_limit | 0 (unlimited) |

It is also possible to set submit number limit per individually designated group/user name, please refer to 11. Limit per Group and User.

Submit limits set on queues can be confirmed by qstat(1) command -Q, -f option as follows.(Please refer to 4.5.2. Interactive Queue.)

- The number of requests that can be submitted to a queue: Submit Number Limit
- The number of requests that one user can submit to a queue: Submit User Number Limit
- The number of requests that one group can submit to a queue: Submit Group Number Limit

(For each item, limit values set are shown or if no limits are set "UNLIMITED" is shown.)

If both submit limit per batch system and submit limit per queue are set, requests cannot be submitted at the point of exceeding either limit.

(6) Suspend Privilege

By setting a suspend privilege to the interactive queue, only a suspend request issued by user with higher privilege than the privilege that is set for the queue can be accepted. Set the suspend privilege per interactive queue using the "set interactive_queue suspend_privilege" sub-command of the qmgr(1M) command.
An example of setting the suspend privilege as higher than operator privilege is shown below.

```
$ qmgr -Po
Mgr: set interactive_queue suspend_privilege = operator inter1
Set Suspend Privilege.queue: inter1
```

To delete the suspend privilege, execute the "delete interactive_queue suspend_privilege" sub-command of the qmgr(1M) command.

(7) User EXIT Script

Set the user exit file per interactive queue using the "set interactive_queue userexit" sub-command of the qmgr(1M) command.

An example of setting the script file script1 that is to be executed at pre-running state for requests of the interactive queue "inter1" is shown below.

```
$ qmgr -Po
Mgr: set interactive_queue userexit location = pre-running script = (script1) queue
= inter1
Set Userexit Script: inter1
```

Execute the "delete interactive_queue userexit" sub-command of the qmgr(1M) command to delete the user exit script setting.

An example of deleting the specified script of the interactive queue "inter1" to run at pre-running is shown below.

```
$ qmgr -Po
Mgr: delete interactive_queue userexit location = pre-running queue = inter1
Delete Userexit Script: inter1
```

(8) Setting Time-out of the User EXIT Execution

When set time-out time of the User EXIT execution, and the User EXIT occurred any trouble like a stall, it is possible to send a KILL signal to the User EXIT and abort the execution. Time-out time is set by the sub-command of qmgr (1M).

An example of setting time-out time of the User EXIT execution to 900 seconds to interactive queue "inter1" is shown below.

```
$ qmgr -Po
Mgr: set interactive_queue userexit_timeout = 900 inter1
Set UserExit Timeout: inter1
```

A default value of time-out time of the User EXIT execution is 0 (off). In case of 0, time-out is not occurred.

(9) Limitation of the Job Number

It is possible to limit the number of jobs created from request. Set limitation of the number of jobs per interactive queue using the "set interactive_queue jobs_range" sub-command of the qmgr(1M) command.

An example of setting a range of 10 to 50 for the number of jobs in the interactive queue "inter1" is shown below.

```
$ qmgr -Po
Mgr: set interactive_queue jobs_range = (10,50)  inter1
Set Jobs Range: inter1
```

It is also possible to set limitation of the job number per individually designated group/user name, please refer to 11. Limit per Group and User.

(10) Waiting Option

If there are enough execution hosts to be assigned immediately, it is possible to select whether wait for enough execution hosts or cancel the request.

Set wait mode using the "set interactive_queue real_time_scheduling" sub-command of the qmgr(1M) command.

An example of setting wait mode in the interactive queue "inter1" is shown below.

```
$ qmgr -Po
Mgr: set interactive_queue real_time_scheduling = wait inter1
Set Real Time Scheduling.queue: inter1
```

Wait mode can be set as follows.

| wait mode | Description |
|---|---|
| wait | Wait for execution host available. |
| submit_cancel | Cancel the submitted request. |
| manual | Depend on run-time option of the submit command (qlogin -W) |

(11) Compulsion Execution Shell

A login shell is usually used as a login shell for interactive queue session. However, when a compulsion execution shell is set as interactive queue, the shell is used as a login shell for interactive queue session.

Set compulsion execution shell using the "set interactive_queue restrict_shell" sub-command of the qmgr(1M) command

An example of setting /bin/shell for restrict shell in the interactive queue "inter1" is shown below.

```
$ qmgr -Po
Mgr: set interactive_queue restrict_shell = /bin/bash inter1
Set Restrict Shell. queue: inter1
```

To delete the setting, use the "delete interactive_queue restrict_shell" sub-command of qmgr(1M).

(12) Idle Timer

A login shell for interactive queue session terminates after waiting for a certain time set as idle timer if input does not arrive. It is possible to set an idle timer every interactive queue.

An example of setting 300 for idle timer in the interactive queue "inter1" is shown below.

```
$ qmgr -Po
Mgr: set interactive_queue idle_timer = 300 inter1
Set idle_timer. inter1
```

When the idle timer is set as 0, which is default setting, the idle timer function is not available.

> [Notes]
> In case MPI job (qlogin -T *MPI-job*), idle timer is not available.

(13) Allowance for exclusive execution request

It is able to configure the allowance for exclusive execution request (which is submitted by qlogin --exclusive) to the queue. The exclusive execution request always executed as 1 logical host(job) per 1 host and it ignore the resource limit (CPU, MEMORY, Virtual MEMORY, GPU, VE) for logical host of the queue. Therefore this configuration can limit the submission of that request.

Allowance for exclusive execution request for interactive queue can be set by following qmgr sub-command with operator privilege.

```
set interactive_queue exclusive_submit {on | off} <queue>
```

off rejects the submission of the request with --exclusive option. on allows the submission of the request with --exclusive option. The default is off.

## 4.3. Routing Queue

### 4.3.1. Create Routing Queue

Routing Queue is the queue that controls the routing of the batch requests. Requests submitted to the routing queue are forwarded to the forwarding queue automatically.

Creates a new routing queue using the "create routing_queue" sub-command of the qmgr(1M) command.

```
$ qmgr -Pm
Mgr: create routing_queue = route1 priority = 20
```

Routing Queue "route1" will be created with above procedure. The priority specified here is the queue priority.

### 4.3.2. Routing Queue Configuration

(1) Queue Priority

The Queue Priority is the priority among queues, and requests of queue with larger values will

be transferred first.

This attribute must always be defined when a queue is created. It can be changed later by the "set routing_queue priority" sub-command of qmgr(1M) command.

```
$ qmgr -Po
Mgr: set routing_queue priority = 10 route1
```

The queue priority of Routing Queue "route1" is changed to 10 with above procedure.

(2) Submit Limit

It is possible to set a limit to the number of batch requests that can be submitted to a routing queue. The number of batch requests that can be submitted means the total number of batch requests existent in a routing queue at the same time. With this submit limit per routing queue, more detailed submit limit of requests can be configured than submit limit per batch system. (For the details of submit limit per batch system, please refer to 2.3.3. Submit Limit.)

The submit limit per routing queue can also be specified per whole queue, group and user. Specify the limit to the number of requests using sub-commands of qmgr(1M). The operator privilege or higher is necessary to set this limit.

The sub-commands and the default values are as follows.

| Attribute | qmgr (1M) sub-command | Default |
|---|---|---|
| The number of requests which can be submitted to a routing queue | set routing_queue submit_limit | 0 (unlimited) |
| The number of requests which one group can submit to a routing queue | set routing_queue group_submit_limit | 0 (unlimited) |
| The number of requests which one user can submit to a routing queue | set routing_queue user_submit_limit | 0 (unlimited) |

It is also possible to set submit number limit per individually designated group/user name, please refer to 11. Limit per Group and User.

Submit limits set on queues can be confirmed by qstat(1) command -Q, -f option as follows.(Please refer to 4.5.3. Routing Queue.)

- The number of requests that can be submitted to a queue: Submit Number Limit
- The number of requests that one user can submit to a queue: Submit User Number Limit
- The number of requests that one group can submit to a queue: Submit Group Number Limit

(For each item, limit values set are shown or if no limits are set "UNLIMITED" is shown.)

If both submit limit per batch system and submit limit per queue are set, requests cannot be submitted at the point of exceeding either limit.

(3) Routing Queue Run Limit

The Routing Queue Run Limit is the maximum number of batch requests that can be routed in a routing queue, simultaneously.

This attribute can be defined at creating a new queue. Note that it can be changed later by the "set routing_queue run_limit" sub-command of qmgr(1M) command.

```
$ qmgr -Po
Mgr: set routing_queue run_limit = 10 route1
```

The run limit of Routing Queue "route1" is changed to "10" with above procedure.

Definable max value is 100.

(4) Destination of Routing Queue

As a destination of routing queues, only batch queue can be specified. Interactive queue is not effective as destination of routing queue.

Multiple queues can be defined as destinations of the routing queue. When multiple queues are set, requests are tried to be forwarded to the queues in turn until the forwarding succeed.

This attribute can be defined when a routing queue is created, but can also be changed or added later.

**Definition and Change of Forwarding**

Define or change routing queues by the "set routing_queue destination" sub-command of qmgr(1M) command.

```
$ qmgr -Po
Mgr: set routing_queue destination = (batch1@host1) route1
```

The queue "batch1" on the batch server host "host1" will be defined as the routing queue "route1" with above procedure. If the routing queue is set already, the setting is overwritten.

If the routing queue is on the same batch server, it is possible to omit a part of "@host1".

**Adding Forwarding**

Add routing queues by the "add routing_queue destination" sub-command of qmgr(1M) command.

```
$ qmgr -Po
Mgr: add routing_queue destination = (batch2@host1) route1
```

The queue "batch2" on the batch server host "host1" will be added to the routing queue "route1" with above procedure. "batch1" and "batch2" will be the routing queue of "route1" if batch1 is already defined as the routing queue of "route1".

**Delete Forwarding**

Delete routing queues by the "delete routing_queue destination" sub-command of qmgr(1M)

command.

```
$ qmgr -Po
Mgr: delete routing_queue destination = (batch2@host1) route1
```

The queue "batch2" on the batch server host "host1" will be deleted from destination of routing queue "route1" with above procedure.

(5) Waiting Interval

If node of the routing queues can accept requests, routing will be retried again at the regular time intervals. It is possible to set per batch server. (Please refer to 2.3.5. Routing Retry Interval.)

## 4.4. Network Queue

### 4.4.1. Create Network Queue

Network Queue is the queue used for file-staging between the client hosts and the batch server host, or the execution host, used by NQSV system. ( About file staging., please refer to 5.1.3. File Staging ). When transmit parameters need to be changed or external file-staging is used, a network queue must be created. The default network queue (DefaultNetQue) is used when the corresponding network queue does not exist.

Create a new network queue using the "create network_queue" sub-command of qmgr(1M) command.

```
$ qmgr -Pm
Mgr: create network_queue = net1 staging_machine = host1 priority = 20
```

Network Queue "net1" for the client host "host1" will be created with above procedure. The priority specified here is the queue priority.

### 4.4.2. Network Queue Configuration

(1) Queue Priority

The Queue Priority is the priority among queues, it puts priorities for file-staging.

This attribute must be defined when a queue is created. Note that it can be changed later by the "set network_queue priority" sub-command of qmgr(1M) command.

```
$ qmgr -Po
Mgr: set network_queue priority = 10 net1
```

The queue priority of network queue "net1" is changed to "10" with above procedure.

(2) Network Queue Run Limit

The Network Queue Run Limit is the maximum number of network requests that can be routed in a network queue simultaneously.

The network queue run limit can be defined when a queue is created. Note that it can also be

changed later by the "set network_queue run_limit" sub-command of qmgr(1M) command.

```
$ qmgr -Po
Mgr: set network_queue run_limit = 10 net1
```

The run limit of network queue "net1" is changed to 10 with above procedure.

Definable max value is 1000.

(3) Run Limit for Each Request

It is possible to set the network queue run limit for each request.

Change the value using the "set network_queue batch_request_run_limit" sub-command of qmgr(1M) command.

```
$ qmgr -Po
Mgr: set network_queue batch_request_run_limit = 10 net1
```

The run limit for each request of network queue "net1" is changed to 10 with above procedure.

(4) Forwarding Host

The forwarding host is is actually client hosts that are the destination of staging of the network queue. This attribute can be defined when a network queue is created, but can also be changed later.

Change the forwarding host of network queues by the "set network_queue staging_machine" sub-command of qmgr(1M) command.

```
$ qmgr -Po
Mgr: set network_queue staging_machine = host1 net1
```

The forwarding host of network queue "net1" is changed to host1.

(5) Switching the Staging Method

There are two type methods for file-staging. One is internal staging method, the other is external staging method.

Switch the staging method by the "set network_queue staging_method" sub-command of qmgr(1M) command.

```
$ qmgr -Po
Mgr: set network_queue staging_method = external net1
```

The staging method of network queue "net1" is changed to the external staging method with above procedure.

A default value is internal file-staging method. ( about staging method, please refer to 5.1.3. File Staging.)

(6) Change of Transfer Buffer Size

Change the transfer buffer size for internal staging method by the "set network_queue staging_extended_buffer_size" sub-command of qmgr(1M) command. The default value is 4 K

bytes.

```
$ qmgr -Po
Mgr: set network_queue staging_extended_buffer_size = 128 net1
```

The transfer buffer size of network queue "net1" is changed to 128 Kbytes with above procedure.

(7) Waiting Interval

If file transfer cannot be done, it will be retried again at the regular time intervals. It is possible to set per batch server. (Please refer to 2.3.5.Routing Retry Interval.)

## 4.5. Queue Information

### 4.5.1. Batch queue

The queue status can be checked using the qstat(1) command with -Q option. To check the status of the batch queues, execute with -e option.

```
$ qstat -Q -e
[EXECUTION QUEUE] Batch Server Host: host1
========================================
QueueName       SCH JSVs ENA STS  PRI TOT ARR WAI QUE PRR RUN POR EXT HLD HOL RST SUS MIG STG CHK
--------------- --- ---- ------- ---- ---------------------------------------------------------------
batch1            0   64 ENA ACT   10   1   0   0   0   0   1   0   0   0   0   0   0   0   0   0
batch2            0   32 ENA ACT   20   3   0   0   2   0   1   0   0   0   0   0   0   0   0   0
--------------- --- ---- ------- ---- ---------------------------------------------------------------
<TOTAL>                                4   0   0   2   0   2   0   0   0   0   0   0   0   0   0
--------------- --- ---- ------- ---- ---------------------------------------------------------------
```

Column "SCH" displays the schedule number, column "JSVs" displays the number of job server linkup and bound, column "ENA" and "STS" displays status of queue, column "TOT" displays the number of submitted requests, and subsequent columns display numbers of requests for each state.

Adding -t option to qstat -Q -e, column "JSVs" displays the number of job server bound including both of linkup and linkdown.

The status of an individual queue can be also checked if the queue name is specified with qstat(1) command as an argument.

```
$ qstat -Q batch1
[EXECUTION QUEUE] Batch Server Host: host1
========================================
QueueName       SCH JSVs ENA STS  PRI TOT ARR WAI QUE PRR RUN POR EXT HLD HOL RST SUS MIG STG CHK
--------------- --- ---- ------- ---- ---------------------------------------------------------------
batch1            0   64 ENA ACT   10   1   0   0   0   0   1   0   0   0   0   0   0   0   0   0
--------------- --- ---- ------- ---- ---------------------------------------------------------------
<TOTAL>                                1   0   0   0   0   1   0   0   0   0   0   0   0   0   0
--------------- --- ---- ------- ---- ---------------------------------------------------------------
```

More detailed information of a queue can be shown using the qstat(1) command with -Q, -f option.

```
$ qstat -Q -f batch1
Execution Queue: batch1@host1
    Run State    = Active
    Submit State = Enable
    Scheduler ID   = 1
    Scheduler Name = Scheduler0001
    Job Server Number = {
       412*  414*
    }
    Node Group Name = {
      (UNBIND)
    }
    Use Supplementary Groups ID for Access Privileges Check = OFF
    Noaccess User list  = {
      (none)
    }
    Noaccess Group list  = {
      (none)
    }
    Refuse submission = {
      (none)
    }
    NUMA Control = OFF
    Hold Privilege    = (none)
    Suspend Privilege = (none)
    Queue Priority = 10
    Submit Number Limit       = UNLIMITED
    Submit User Number Limit  = UNLIMITED
    Submit Group Number Limit = UNLIMITED
    Subrequest Number Limit = 1000
    Checkpoint Interval = 0
    Auto Bind JobServer = ON
    Restart File Directory = /var/opt/nec/nqsv/jsv/rstfdir
    qattach command = Enable
    IntelMPI Process Manager = hydra
    Hook Function = OFF
    Exclusive Submit = ON
    UserExit Timeout = OFF
    UserPP Timeout = 300
    File Stageout = ON
    Delete Failed Urgent Request = OFF
    Partial Process Swapping = ON
    GPU-CPU Affinity = OFF
    GPU-CPU Affinity CPU per GPU = 1
    NEC MPI Process Manager = mpd
    Range of Jobs Limit per Batch Request (min,max) = 1,10240
    Range of Request Priority = {
      manager     (min,max) = -1024,1023
      operator    (min,max) = -1024,1023
      specialuser (min,max) = -1024,1023
      user        (min,max) = -1024,1023
    }
    Defined VE Number     = 1
    Submit VE Node Range (min,max) = 0,UNLIMITED
    Total Request         = 0
    Arriving Request      = 0
    Waiting Request       = 0
    Queued Request        = 0
    Pre-running Request   = 0
```

```
    Running Request       = 0
    Post-running Request  = 0
    Exiting Request       = 0
    Held Request          = 0
    Holding Request       = 0
    Restarting Request    = 0
    Suspending Request    = 0
    Suspended Request     = 0
    Resuming Request      = 0
    Migrating Request     = 0
    Staging Request       = 0
    Checkpointing Request = 0
  UserExit Script:
    (none)
  Logical Host Resource Ranges:
    VE Node Number        = Min:         0 Max: UNLIMITED Warn:       --- Std:         0
    CPU Number            = Min:         1 Max: UNLIMITED Warn:       --- Std:         1
    GPU Number            = Min:         0 Max: UNLIMITED Warn:       --- Std:         0
    CPU Time              = Min:        0S Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    Memory Size           = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    Virtual Memory Size   = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    VE CPU Time           = Min:        0S Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    VE Memory Size        = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    Stdout Size           = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    Stderr Size           = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  Logical Host HCA Resource Ranges:
    For I/O               = Min:      0 Max: UNLIMITED Std:         0
    For MPI               = Min:      0 Max: UNLIMITED Std:         0
    For ALL               = Min:      0 Max: UNLIMITED Std:         0
  VE Node Resource Ranges:
    VE CPU Time           = Min:        0S Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    VE Memory Size        = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  Resources Limits:
    (Per-Req) Elapse Time Limit       = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Job) CPU Time                = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Job) CPU Number              = Max: UNLIMITED Warn:       --- Std:         1
    (Per-Job) Memory Size             = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Job) Virtual Memory Size     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Job) GPU Number              = Max: UNLIMITED Warn:       --- Std:         0
    (Per-Prc) CPU Time                = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Prc) Open File Number        = Max: UNLIMITED Warn:       --- Std: UNLIMITED
    (Per-Prc) Virtual Memory Size     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Prc) Data Segment Size       = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Prc) Stack Segment Size      = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Prc) Core File Size          = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Prc) Permanent File Size     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Prc) VE CPU Time             = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Prc) VE Memory Size          = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  Kernel Parameter:
    Resource Sharing Group     = 0
    Nice Value                 = 0
```

The number in "Job Server Number = { }" means job server number. The number with an asterisk (*) on the right side means that the job server is linkup.

### 4.5.2. Interactive Queue

The queue status can be checked using the qstat(1) command with -Q option. To check the

status of the interactive queues, execute with -i option.

```
$ qstat -Q -i
[INTERACTIVE QUEUE] Batch Server Host: host1
==========================================
QueueName       SCH JSVs ENA STS  PRI  TOT ARR WAI QUE PRR RUN POR EXT HLD SUS
--------------- --- ---- ------- ---- ---------------------------------------
inter1            1    4 ENA ACT   10    1   0   0   0   0   1   0   0   0   0
--------------- --- ---- ------- ---- ---------------------------------------
<TOTAL>                                 1   0   0   0   0   1   0   0   0   0
--------------- --- ---- ------- ---- ---------------------------------------
```

More detailed information of a queue can be shown using the qstat(1) command with -Q, -f option. And the status of an individual queue can be also checked if the queue name is specified with qstat(1) command as an argument.

```
$ qstat -Q -f inter1
Interactive Queue: inter1@host1
    Run State    = Active
    Submit State = Enable
    Scheduler ID   = 1
    Scheduler Name = Scheduler0001
    Job Server Number = {
      414*
    }
    Node Group Name = {
      (UNBIND)
    }
    Use Supplementary Groups ID for Access Privileges Check = OFF
    Noaccess User list  = {
      (none)
    }
    Noaccess Group list = {
      (none)
    }
    Refuse submission = {
      (none)
    }
    NUMA Control = OFF
    Suspend Privilege = (none)
    Queue Priority = 10
    Submit Number Limit       = UNLIMITED
    Submit User Number Limit  = UNLIMITED
    Submit Group Number Limit = UNLIMITED
    Auto Bind JobServer = ON
    qattach command = Enable
    IntelMPI Process Manager = hydra
    Hook Function = OFF
    UserExit Timeout = OFF
    UserPP Timeout = 300
    Exclusive Submit = ON
    Range of Jobs Limit per Interactive Request (min,max) = 1,10240
    Real Time Scheduling = wait
```

```
  Idle Timer = 0
  Restrict Shell = (none)
  Defined VE Number     = 1
  Submit VE Node Range (min,max) = 0,UNLIMITED
  Total Request        = 0
  Arriving Request     = 0
  Waiting Request      = 0
  Queued Request       = 0
  Pre-running Request  = 0
  Running Request      = 0
  Post-running Request = 0
  Exiting Request      = 0
  Held Request         = 0
  Suspending Request   = 0
  Suspended Request    = 0
  Resuming Request     = 0
UserExit Script:
  (none)
Logical Host Resource Ranges:
  VE Node Number        = Min:         0 Max: UNLIMITED Warn:       --- Std:         0
  CPU Number            = Min:         1 Max: UNLIMITED Warn:       --- Std:         1
  GPU Number            = Min:         0 Max: UNLIMITED Warn:       --- Std:         0
  CPU Time              = Min:        0S Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  Memory Size           = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  Virtual Memory Size   = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  VE CPU Time           = Min:        0S Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  VE Memory Size        = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
Logical Host HCA Resource Ranges:
  For I/O               = Min:      0 Max: UNLIMITED Std:         0
  For MPI               = Min:      0 Max: UNLIMITED Std:         0
  For ALL               = Min:      0 Max: UNLIMITED Std:         0
VE Node Resource Ranges:
  VE CPU Time           = Min:        0S Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  VE Memory Size        = Min:        0B Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
Resources Limits:
  (Per-Req) Elapse Time Limit       = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) CPU Time                = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) CPU Number              = Max: UNLIMITED Warn:       --- Std:         1
  (Per-Job) Memory Size             = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) Virtual Memory Size     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Job) GPU Number              = Max: UNLIMITED Warn:       --- Std:         0
  (Per-Prc) CPU Time                = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) Open File Number        = Max: UNLIMITED Warn:       --- Std: UNLIMITED
  (Per-Prc) Virtual Memory Size     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) Data Segment Size       = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) Stack Segment Size      = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) Core File Size          = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) Permanent File Size     = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) VE CPU Time             = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  (Per-Prc) VE Memory Size          = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
Kernel Parameter:
  Resource Sharing Group     = 0
  Nice Value                 = 0
```

### 4.5.3. Routing Queue

The status of routing queues can be also checked using -Q option of the qstat(1) command. To check the information of only routing queues, execute with -r option.

```
$ qstat -Q -r
[ROUTING QUEUE] Batch Server Host: host1
========================================
QueueName       ENA STS PRI RLM  TOT ARR WAI QUE HLD TRS
--------------- --------------- -------------------------
pipe1           ENA INA  20   1    2   2   0   0   0   0
pipe2           DIS INA  30   1    0   0   0   0   0   0
netpipe1        ENA ACT  20   1    2   1   1   0   0   0
--------------- --------------- -------------------------
<TOTAL>                     100    4   3   1   0   0   0
--------------- --------------- -------------------------
```

More detailed information of a queue can be shown using the qstat(1) command with -Q, -f option. And the status of an individual queue can be also checked if the queue name is specified with qstat(1) command as an argument.

```
$ qstat -Q -f pipe1
Routing Queue: pipe1@host1
    Run State    = Inactive
    Submit State = Enable
    Queue Priority = 20
    Total Run Limit = 1
    Submit Number Limit = UNLIMITED
    Submit User Number Limit  = UNLIMITED
    Submit Group Number Limit = UNLIMITED
    Total Request      = 2
    Arriving Request   = 2
    Waiting Request    = 0
    Queued Request     = 0
    Held Request       = 0
    Transiting Request = 0
    Destinations = {
      batch1@host1
      batch2@host1
    }
    Use Supplementary Groups for Access Privileges Check = OFF
    Noaccess User list = {
      (none)
    }
    Noaccess Group list = {
      (none)
    }
    Refuse submission = {
      (none)
    }
```

### 4.5.4. Network Queue

The status of network queues can be also checked using -Q option of the qstat(1) command. To check the information of only network queues, execute qstat with -N option.

```
$ qstat -Q -N
[NETWORK QUEUE] Batch Server Host: host1
========================================
QueueName       StagingMachine       ENA STS PRI RLM  TOT WAI QUE RUN
--------------- -------------------- --------------- ----------------
DefaultNetQue    (any)                ENA ACT   0  1    0   0   0   0
net1            client1               ENA ACT  10  1    0   0   0   0
--------------- -------------------- --------------- ----------------
<TOTAL>                                          1000    0   0   0   0
--------------- -------------------- --------------- ----------------
```

More detailed information of a queue can be shown using the qstat(1) command with -Q, -f option. And the status of an individual queue can be also checked if the queue name is specified with qstat(1) command as an argument.

```
$ qstat -Q -f net1
Network Queue: net1@host1
    Run State    = Active
    Submit State = Enable
    Queue Priority = 10
    Total Run Limit = 1
    Run Limit per Batch Request = 1
    Staging Machine = client1
    Staging Method  = Internal
    Staging Extended Buffer = 512KB
    Total Request   = 0
    Waiting Request = 0
    Queued Request  = 0
    Running Request = 0
```

### 4.5.5. Customizing Information

It is possible to specify items to be displayed with -F option of qstat(1) command. At this time, -e for batch queue or -i for interactive queue or -r for routing queue or -N for network queue is needed. It is not possible to specify items with -f option for detail information.

```
$ qstat -Q -e -F quenm,stt1,stt2,qtot
[EXECUTION QUEUE] Batch Server Host: host1
==========================================
QueueName       ENA STS  TOT
--------------- --- --- ----
batch1          ENA ACT    1
batch2          ENA ACT    3
--------------- --- --- ----
<TOTAL>                    4
```

```
--------------- --- --- ----
```

Items that can be specified for queue information are as follows.

| Item | content | Summary display format |
|---|---|---|
| Batch queue (-Q [-e]) | | |
| quenm | Queue name | QueueName |
| schid | Scheduler ID bound to the queue | SCH |
| jsvs | Number of job servers bound to the queue | JSVs |
| stt1 | Request acceptable or not | ENA |
| stt2 | Request executable or not | STS |
| pri | Queue priority | PRI |
| qtot | Number of batch requests submitted in a queue | TOT |
| arr | Number of batch requests for each state | ARR |
| wai | | WAI |
| que | | QUE |
| prr | | PRR |
| run | | RUN |
| por | | POR |
| ext | | EXT |
| hld | | HLD |
| hol | | HOL |
| rst | | RST |
| sus | | SUS |
| mig | | MIG |
| stg | | STG |
| chk | | CHK |
| ehost* | Execution host bound to the queue | ExecutionHost |
| attbl* | Attach request Possible/Not Possible | ATTBL |
| Interactive queue (-Q -i) | | |
| quenm | Queue name | QueueName |
| schid | Scheduler ID bound to the queue | SCH |
| jsvs | Number of job servers bound to the queue | JSVs |
| stt1 | Request acceptable or not | ENA |
| stt2 | Request executable or not | STS |
| pri | Queue priority | PRI |
| qtot | Number of interactive requests submitted in a queue | TOT |

| | | |
|---|---|---|
| arr | Number of interactive requests for each state | ARR |
| wai | | WAI |
| que | | QUE |
| prr | | PRR |
| run | | RUN |
| por | | POR |
| ext | | EXT |
| hld | | HLD |
| sus | | SUS |
| ehost* | Execution host bound to the queue | ExecutionHost |
| attbl* | Attach request Possible/Not Possible | ATTBL |
| Routing queue (-Q -r) | | |
| quenm | Queue name | QueueName |
| stt1 | Request acceptable or not | ENA |
| stt2 | Request executable or not | STS |
| pri | Queue priority | PRI |
| rqrlm | Run limit for each queue | RLM |
| qtot | Number of batch requests submitted in a queue | TOT |
| arr | Number of batch requests for each state | ARR |
| wai | | WAI |
| que | | QUE |
| hld | | HLD |
| trs | | TRS |
| Network queue (-Q -N) | | |
| quenm | Queue name | QueueName |
| stghost | Staging host name | StagingMachine |
| stt1 | Request acceptable or not | ENA |
| stt2 | Request executable or not | STS |
| pri. | Queue priority | PRI |
| rqrlm | Run limit for each queue | RLM |
| qtot | Number of batch requests submitted in a queue | TOT |
| wai | Number of batch requests for each state | WAI |
| que | | QUE |
| run | | RUN |

***** This item can be specified in case of displaying information using qstat(1) command -F option. (About details of qstat(1) command's -F option, please refer to [Operation]Customizing Information.)

It is also possible to sort information by any item using -o option or -O option of qstat(1) command.(About qstat(1) -O, and -o options, please refer to [Operation]Sorting Information.)

### 4.6. Queue State

Queue has two types of state. One is whether the queue can accept a request submitted or not, the other is whether the queue can execute a request or not. These states are defined as follows.

| State | value | description |
|---|---|---|
| Request acceptable or not | ENABLE | The queue can accept a request submitted |
| | DISABLE | The queue cannot accept a request submitted |
| Request executable or not | ACTIVE | The queue can execute a request |
| | INACTIVE | The queue cannot execute a request |

#### 4.6.1. Enable/Disable State

Default values of each state are the DISABLE state and the INACTIVE state. The status of a queue can be set to accept requests by executing "enable" sub-command of qmgr(1M) command.

| queue | qmgr(1M) sub-command |
|---|---|
| Execution queue | enable execution_queue |
| Interactive queue | enable interactive_queue |
| Routing queue | enable routing_queue |
| Network queue | enable network_queue |

An execution queue "batch1" is set to accept requests as follows. The operator privilege is necessary.

```
$ qmgr -Po
Mgr: enable execution_queue = batch1
Enable Queue: batch1
```

The status of a queue can be set to refuse batch requests by "disable" sub-command of qmgr(1M) command.

| queue | qmgr(1M) sub-command |
|---|---|
| Execution queue | disable execution_queue |
| Interactive queue | disable interactive_queue |
| Routing queue | disable routing_queue |
| Network queue | disable network_queue |

An execution queue "batch1" is set to refuse submission of requests as follows. The operator privilege is necessary.

```
$ qmgr -Po
Mgr: disable execution_queue = batch1
Disable Queue: batch1
```

### 4.6.2. Active/Inactive State

The status of a queue can be set to execute requests by "start" sub-command of qmgr(1M) command.

| queue | qmgr(1M) sub-command |
|---|---|
| Execution queue | start execution_queue |
| Interactive queue | start interactive_queue |
| Routing queue | start routing_queue |
| Network queue | start network_queue |

An execution queue "batch1" is set to allow execution of requests as follows. The operator privilege is necessary.

```
$ qmgr -Po
Mgr: start execution_queue = batch1
Start Queue: batch1
```

It is possible to change all queue, which are execution queue or interactive queue or routing queue or network queue, to allow execution ("ACTIVE") state by "start all queue" sub-command of qmgr(1M).

The status of a queue can be set to disable executing requests by stop sub-command of qmgr(1M) command.

| queue | qmgr(1M) sub-command |
|---|---|
| Execution queue | stop execution_queue |
| Interactive queue | stop interactive_queue |
| Routing queue | stop routing_queue |
| Network queue | stop network_queue |

An execution queue "batch1" is set to disable execution of requests as follows. The operator privilege is necessary.

```
$ qmgr -Po
Mgr: stop execution_queue = batch1
Stop Queue: batch1
```

It is possible to change all queue, which are execution queue or interactive queue or routing queue or network queue, to disable execution ("INACTIVE") state by "stop all queue" sub-command of qmgr(1M).

### 4.7. Bind to Job Server and Scheduler

Note that execution queues and interactive queue must be bound to the job server, which actually execute requests, and to the scheduler, which schedules execution of requests, in order to execute requests. The sub-commands of qmgr are as follows.

| queue | binding job server | binding scheduler |
|---|---|---|
| Execution queue | bind execution_queue job_server | bind execution_queue scheduler |
| Interactive queue | bind interactive_queue job_server | bind interactive_queue scheduler |

An execution queue "batch1" and job server "100" are bound as follows.

```
$ qmgr -Pm
Mgr: bind execution_queue job_server batch1 job_server_id = 100
Bound Job_Server_ID (100)  to Queue (batch1).
```

An interactive queue "inter1" and scheduler "1" are bound as follows.

```
$ qmgr -Pm
Mgr: bind interactive_queue scheduler inter1 scheduler_id = 1
Bound Scheduler_ID (1) to Queue (inter1).
```

It is possible to unbind the job server and the scheduler. The sub-commands of qmgr are as follows.

| queue | Unbinding job server | Unbinding scheduler |
|---|---|---|
| Execution queue | unbind execution_queue job_server | unbind execution_queue scheduler |
| Interactive queue | unbind interactive_queue job_server | unbind interactive_queue scheduler |

The job server "100" is unbound from an execution queue "batch1" as follows.

```
$ qmgr - Pm
Mgr: unbind execution_queue job_server batch1 job_server_id = 100
Unbound Job_Server_ID (100) to Queue (batch1).
```

The scheduler is unbound from interactive queue "inter1" as follows.

```
$ qmgr -Pm
Mgr: unbind interactive_queue scheduler inter1
Unbound Scheduler_ID (1) to Queue (inter1).
```

## 4.8. Queue Access Limit Set

Set the access limit for a batch queue, interactive queue and routing queue to limit submitting requests by specific user, group, and route.

### 4.8.1. Access Limitation to User and Group

It is possible to limit submitting requests by specific user and group. Make a list of users/groups who can access a queue, or make a list of users/groups who cannot access a queue.

<u>Access permit setting</u>

Set ACCESS mode to a queue, and then make a list of users/groups that can access a queue using sub-command of qmgr(1M) command. The sub-commands of qmgr(1M) are as follows.

| queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | set execution_queue access |
| Interactive queue | set interactive_queue access |
| Routing queue | set routing_queue access |

Set ACCESS mode to a batch queue "batch1" as follows. The manager privilege is necessary.

```
$ qmgr -Pm
Mgr: set execution_queue access batch1
Set Access_List Access.
```

When the access mode of a queue is changed to ACCESS mode, the root is registered to a user access list as defaults, and no group is registered to a group access list. Only the root user can access the queue.

Then, set the users that can access the queue to the access list. The following sub-commands of qmgr(1M) are used for this setting.

| Setting | sub-command |
|---|---|
| A list of the user names is set to the access list. | set execution_queue users<br>set interactive_queue users<br>set routing_queue users |
| A user is added to the access list. | add execution_queue users<br>add interactive_queue users<br>add routing_queue users |
| A user is deleted from the access list. | delete execution_queue users |

| | |
|---|---|
| | delete interactive_queue users<br>delete routing_queue users |

The user "user1" is added to the access list of batch queue "batch1" as follows.

```
$ qmgr -Pm
Mgr: add execution_queue users = user1 batch1
Set user list.queue: batch1
```

Set the groups that can access the queue to the access list. The following sub-commands of qmgr(1M) are used for this setting.

| Setting operation | sub-command |
|---|---|
| A list of the group names is set to the access list. | set execution_queue groups<br>set interactive_queue groups<br>set routing_queue groups |
| A group is added to the access list. | add execution_queue groups<br>add interactive_queue groups<br>add routing_queue groups |
| A group is deleted from an access list. | delete execution_queue groups<br>delete interactive_queue groups<br>delete routing_queue groups |

The group "group1" is added to the access list of batch queue "batch1" as follows.

```
$ qmgr -Pm
Mgr: add execution_queue groups = group1 batch1
Set group list.queue: batch1.
```

Access permit setting information is displayed by qstat -Qf as follows.

```
$ qstat - Qf
      :
    Access User list  = {
      root    user1
    }
    Access Group list = {
      group1
    }
      :
```

<u>Access denied setting</u>

Set NOACCESS mode to a queue, and then make a list of users/groups that cannot access a queue.

It is possible to set either ACCESS mode or NOACCES mode to a queue. NOACCESS mode is

default. When a queue is created, its access mode is NOACCESS and access list for users/groups are empty. Therefore, every user/group can access the queue.
The sub-commands of qmgr(1M) are as follows.

| queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | set execution_queue noaccess |
| Interactive queue | set interactive_queue noaccess |
| Routing queue | set routing_queue noaccess |

Set NOACCESS mode to a batch queue "batch1" as follows.

```
$ qmgr -Pm
Mgr: set execution_queue noaccess batch1
Set Access_List Noaccess.
```

When the access mode of a queue is changed to NOACCESS mode, no user/group is registered to an access list. Every user can access the queue.

Then, set the users and groups that cannot access the queue to the access list. This is the same as access permit setting.
Access denied setting information is displayed by qstat -Qf as follows.

```
$ qstat -Qf
      :
    Noaccess User list = {
      user1    user2
    }
    Noaccess Group list = {
      group1
    }
      :
```

### 4.8.2. Access Limit by Supplementary Group

In the above-mentioned setting of the access limitation for group, access permitting and denying are checked by whether the group name that is set in an access group list agrees with the primary group name of the user.
In addition to the primary group name, the supplementary group name can be added to the target of access enable or disable check for the access limitation of the queue.
For this setting, the sub-commands of qmgr(1M) are as follows.

| queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | set execution_queue supplementary_groups_check |
| Interactive queue | set interactive_queue supplementary_groups_check |
| Routing queue | set routing_queue supplementary_groups_check |

The access limitation by supplementary group for the batch queue "batch1" is set valid as follows.

```
$ qmgr -Pm
Mgr: set execution_queue supplementary_groups_check on batch1
Set Supplementary Groups Check ON. queue: batch1
```

### 4.8.3. Access Limit to Submitting Route

There are the following four routes to submit requests to batch queues, interactive queues and routing queues.

- Request submitting by qsub, qlogin and qrsh
- Request submitting by qmove (except interactive queue)
- Request submitting from local routing queue.(transmission from a routing queue of a same batch server. except interactive queue)
- Request submitting from remote routing queue.(transmission from a routing queue of a different batch server. except interactive queue)

It is possible to set refusal of submitting or not to each queue. For this setting, the sub-commands of qmgr(1M) are as follows.

| queue | qmgr(1M) sub-command | |
|---|---|---|
| | set refusal of submitting | delete refusal of submitting |
| Batch queue | set execution_queue refuse_submission | delete execution_queue refuse_submission |
| Interactive queue | set interactive_queue refuse_submission | delete interactive_queue refuse_submission |
| Routing queue | set routing_queue refuse_submission | delete routing_queue refuse_submission |

To permit only from local routing queue for batch queue "batch1", the operation is executed as follows.

```
$ qmgr -Po
Mgr: set execution_queue refuse_submission = (qsub,qmove,remote_routing)  batch1
Set Refuse submission: batch1
```

## 4.9. Queue Abort

Terminating all requests being executed in a batch queue or an interactive queue forcibly is called "Queue abort". The requests with the following states can be aborted.

- RUNNING

- SUSPENDING
- SUSPENDED
- RESUMING

A queue is aborted by the following sub-command of the qmgr (1M) command.

| queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | abort execution_queue |
| Interactive queue | abort interactive_queue |

```
$ qmgr -Pm
Mgr: abort execution_queue = batch1
Delete 1.host1
```

All requests being executed in execution queue "batch1" are abort with above procedure.
In order to abort all users' requests, the manager privilege is necessary. If qmgr(1M) executed with operator privilege or below, only requests that the operator submitted are aborted.

### 4.10. Queue Purge

Deleting all requests in the queuing state in a batch queue or an interactive queue or a routing queue is called "Queue purge". The requests in the following states can be purged.

- QUEUED
- WAITING
- HELD
- STAGING

A queue is purged by the following sub-command of the qmgr (1M) command.

| queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | purge execution_queue |
| Interactive queue | purge interactive_queue |
| Routing queue | purge routing_queue |

```
$ qmgr -Pm
Mgr: purge execution_queue = batch1
Delete 2.host1
```

All requests queued in the batch queue "batch1" are deleted with above procedure
In order to purge all users' requests, the manager privilege is necessary. If qmgr(1M) executed with operator privilege or below, only requests that the operator submitted are purged.

### 4.11. Queue Delete

Delete a queue using the delete sub-command of the qmgr(1M) command. In order to delete a queue, it is necessary that the queue has no request and in the state requests cannot be accepted (DISABLE state).

A queue is deleted by the following sub-command of the qmgr (1M) command.

| queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | delete execution_queue |
| Interactive queue | delete interactive_queue |
| Routing queue | delete routing_queue |
| Network queue | delete network_queue |

```
$ qmgr -Pm
Mgr: delete execution_queue queue = batch1
Queue batch1 deleted.
```

A batch queue "batch1" is deleted with above procedure.

# 5. Request Management

## 5.1. Batch request

### 5.1.1. Request Attribute Change

It is possible to change attributes set to a request by the qalter(1) command. Please refer to the qalter (1) command about changeable attribute.

Even during execution of a request, it is possible to change the kernel parameter attributes. But other attributes need to be changed before beginning execution of a request.

For example, the nice value of the kernel parameter attribute of request 72.host1 is changed to 20 as follows.

```
$ qalter -P o -K nice=20 72.host1
Attribute of Request is altered.
```

### 5.1.2. User EXIT

User EXIT is the function which executes the user-defined shell script (**User EXIT script**) on the execution host where each job is executed when the request transits in the specific state.

The points of timing, or location, when the request transits are as follows, and it is possible to set at most 4 scripts for each timing.

- PRE-RUNNING (Just before starting of execution)
- POST-RUNNING (Just after terminating of execution)

(1) User EXIT setting

The user EXIT is set to a batch queue or interactive queue by the sub-command of following qmgr (1M).

| queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | set execution_queue userexit |
| Interactive queue | set interactive_queue userexit |

Set the timing of the script starting to "location=", set the script file to "script=", set the queue to "queue=".

The keywords that can be set to the timing "location=" are as follows.

| Timing | key word |
|---|---|
| PRE-RUNNING | pre-running |
| POST-RUNNING | post-running |

When at most 4 scripts being set to "script=", they are divided by a comma (,) or a colon (:). When a comma (,) being used, the scripts are executed serially in order from the left. When a colon (,) being used, the scripts are executed at the same time.

It is necessary to put the script to /opt/nec/nqsv/sbin/uex_prog/ directory on the batch server host.

The following is an example of setting the user EXIT script script1 and script2 serially before request's starting to a batch queue "batch1".

```
$ qmgr -Po
Mgr: set execution_queue userexit location = pre-running script = (script1,script2)
queue = batch1
Set Userexit Script:batch1
```

A setting information of the user EXIT is displayed as follows by -Q -f option of the qstat(1) command.

```
 Execution Queue: batch1@host1
     :
   UserExit Script:
     Pre-running = {
       1st = script1
       2nd = script2
     }
     :
```

(2) Execution of the user EXIT

If user EXIT is set in the queue, before executing the user EXIT script, the user EXIT script file put under the /opt/nec/nqsv/sbin/uex_prog/ directory is copied to each job server that executes a job of the request and executed. For example, if "location=" is specified as pre-running, the user EXIT script will be copied to the job server and executed when the request is PRE-RUNNING state.

When more than one user EXIT scripts are set divided by a comma to be executed serially, the execution order of scripts is synchronized among jobs. In other words, after execution of the script set as first has ended in all jobs, execution of the script set as second begins.

When the user EXIT scripts are executed, a value "EXECUTION" is set to environment variable "UEX_PROCTYPE". When the exit code of the user EXIT script is 0, its execution is judged as success and the state of the request moves to the next state.

When the exit code of the user EXIT script is non-zero, the transition of the request state fails. Then, if it is not POST-RUNNING state, the user EXIT scripts are executed in reverse order from the failed scripts with environment variable "UEX_PROCTYPE" being set "ROLLBACK". Therefore, if a rollback processing is needed for state transition failure, it is necessary that user EXIT scripts have both forwarding processing and rollback processing according to the value of the environment variable "UEX_PROCTYPE".

The following environment variable is set at the time of the user EXIT script execution.

| Environment variable name | Description | Variable value |
|---|---|---|

| | | |
|---|---|---|
| UEX_LOCATION | Location from which user EXIT script is started | " PRERUNNING",<br>" POSTRUNNING" |
| UEX_PROCTYPE | Type of processing | " EXECUTION" : Normal Processing<br>" ROLLBACK" : Rollback Processing |
| UEX_ORDERNO | Execution sequence number of User EXIT script | Integer of 0-3 |
| UEX_JOBID | Job ID executing the user EXIT | <job number>:<sequence number>.<host name> |
| UEX_NJOBS | Number of jobs in request | Integer of one or more |
| UEX_USER | Job owner's username | User name |
| UEX_HOME | Job owner's Home directory | Full path name |
| UEX_STGDIR | The top directory of Staging File Storage Location | Full path name |
| UEX_JOBDB | The top directory of Job Database | Full path name |
| UEX_RSTPREV | The previous state of request | " QUEUED"<br>" RUNNING"<br>" SUSPENDING"<br>" SUSPENDED"<br>" RESUMING" |
| UEX_RSTRSON | The reason of the status transition | " RUN" (Run request)<br>" RERUN" (Rerun request)<br>" DELETE" (Delete request)<br>" EXIT" (Exited)<br>" SYSTEM_FAILURE" (System Failure) |
| UEX_SID | Session ID of job | Integer of 1 or larger |
| UEX_START_TIME | Start time of job | Time-based format: YYYY/MM/DD hh:mm:ss |
| UEX_END_TIME | End time of job | Time-based format: YYYY/MM/DD hh:mm:ss |
| The environment variable set in a job | | |

[Script example]

```
#!/bin/sh

LINK_FILE=$UEX_HOME/jobfiles

case $UEX_LOCATION in
PRERUNNING|RESTARTING)
```

```
  case $UEX_PROCTYPE in
  EXECUTION)
    rm $LINK_FILE > /dev/null 2>&1
    ln -s $UEX_STGDIR $LINK_FILE || exit 1
    ;;
  ROLLBACK)
    rm $LINK_FILE > /dev/null 2>&1
    ;;
  esac
  ;;
POSTRUNNING|HOLDING)
  case $UEX_PROCTYPE in
  EXECUTION)
    rm $LINK_FILE > /dev/null 2>&1
    ;;
  ROLLBACK)
    ln -s $UEX_STGDIR $LINK_FILE > /dev/null 2>&1
    ;;
  esac
  ;;
esac

exit 0
```

(3) User EXIT I/O

The messages written in the standard/error output of user EXIT script is output to the log file on the batch server via the job server.

Note as follows regarding the output to log files.

- It is necessary to set the log level 2 or more to output to a log file. Change the log level using the set batch_server log_file sub-command of the qmgr(1M) command.
- The maximum length of one line is 1023 bytes. When length of the message exceeds, it is divided into two or more lines.

The standard inputs of user EXIT script are redirected to /dev/null. File descriptors (larger than 3) will be closed.

(4) Time-out of the User EXIT Execution

NQSV can monitor the time-out of the UserEXIT execution to prepare the trouble like a stall of User EXIT script.

Time-out time of the User EXIT execution is set to a queue. When time-out time set to a queue has passed after the User EXIT begins execution, script execution is interrupted.

Use the following sub-commands of qmgr (1M) to set the time-out of User EXIT execution.

| Sub-command of qmgr(1M) | Default |
|---|---|

| | |
|---|---|
| set execution_queue userexit_timeout<br>set interactive_queue userexit_timeout | 0 (OFF)<br>Not time-out |

### 5.1.3. File Staging

File Staging Function transfers the files related to the job execution between the client host and the execution hosts.

The following two features for each direction are available for the file staging function.

- Stage-in

  It is a type that the file is transferred from the client-host to the execution-host. It is used to put necessary files for job executing on the execution host beforehand.

  The file for stage-in is set by -I option of the qsub (1) command at the time of submitting request.

- Stage-out

  It is a type that the file is transferred from the execution host to the client host. It is used mainly for transferring output files from the batch job to user.

  The file for stage-out is standard out, standard error, and set by -O option of the qsub (1) command at the time of submitting request.

<u>Network queue</u>

For file staging of a request, a file transfer to the client host is executed as a network request in the NQSV system, and a network queue exists as a queue for it.

It is possible to make a network queue every client host that does a file transfer. In addition, staging type explained below, socket buffer and the number of concurrent executions of a file transfer can be set at this time.

There is DefaultNetQueue made as a default for network queue, and if a network queue for staging is not made specifically, this DefaultNetQueue is used.

<u>Staging method</u>

The method of file staging has two types, Internal Staging Method and External Staging Method. A staging method is set to a network queue. The default value of a staging method of a network queue is the Internal Staging Method.

Execute the "set network_queue staging_method" sub-command of the qmgr(1M) to switch the Internal Staging Method and the External Staging Method.

```
$ qmgr -Po
Mgr: set network_queue staging_method = external net1
Set Staging Method. Network_queue: net1
```

<u>Allow the absolute staging file path</u>

Only the external staging, it is possible to set a staging file pass on the execution host by absolute path.
To enable this function, execute the qmgr (1M) sub-command "set batch_server allow_absolute_exepath" with manager privilege.

```
$ qmgr -Pm
Mgr: set batch_server allow_absolute_exepath on
```

[Notes]
This function enables for the external staging only. Absolute path can't be specified for the internal staging.

(1) Internal Staging Method (Default File Staging Method)

Internal Staging Method is the standard file staging method of NQSV. As all file transferring processed are performed by NQSV system in this method, it is not necessary special setting for starting operation. The transfer rate is low as the files are transferred via the batch server host. Therefore, this method is suitable for a small-scale file staging.
The file transfer by internal staging method is divided into the following two sequences.

1. Transfer between client host and batch server host
   The file is transferred between the User Agent on the client host and Staging Server on the batch server host. On the batch server host, the staging file is temporarily stored in the batch server database.
2. Transfer between execution host and batch server host
   The file is transferred by using the TCP connection between the Batch Server and the Job Server. The buffer size used by this transferring can be changed by the "set network_queue staging_extended_buffer_size" sub-command of qmgr(1M). In general, the performance of transferring is improved by enlarging buffer size.

(2) External Staging Method

The External Staging Method is the method of staging that invokes the user-defined staging function (staging script) from NQSV.

- Setting
  Create a network queue and set a staging as outside transmission (external) to the client host that transfers a file transfer. In addition, install the script file under /opt/nec/nqsv/sbin/stg_prog/ on the batch server host. The script name has to be same as the name of target network queue.

- Staging Script

Staging script are executed as a root privilege for each file on the batch server.

At this time, the following environment variable is established to a staging script.

| Environment variable name | Description | Variable value |
|---|---|---|
| STG_DIRECTION | Direction of staging(IN/OUT) | " STAGE_IN" (stage in)<br>" STAGE_OUT" (stage out) |
| STG_ORDERFILE | Staging order file | Full path name |
| STG_STGNO | Staging number | Integer of 0 or more (4 digit 0 padding from the left) |
| STG_REQID | Request ID | <sequence number>.<host name> |
| STG_USER | User name of request owner on batch server host | Character string |
| STG_GROUP | Group name of request owner on batch server host.<br>Or specified group name by submission command in case Designated Group Execution Function ON.<br>(For detail, please refer to 10. Group of Request.) | Character string |
| STG_UID | User ID of request owner on batch server host | Integer of 0 or more |
| STG_GID | Group ID of request owner on batch server host.<br>Or specified group ID by submission command in case Designated Group Execution Function ON.<br>(For detail, please refer to 10. Group of Request.) | Integer of 0 or more |

- Stage Order file

Stage order file is written the information about staging file, which is set to the environment variable STG_ORDERFILE. Staging script refers this file and transfers a file.

Each line defines one transmission, and the format is as follows.

```
stgno jobno cli_user cli_host cli_path exe_user exe_host exe_path
```

| | |
|---|---|
| stgno | Staging number |
| jobno | Job number |
| cli_user | User name on the client host |
| cli_host | Client host name |
| cli_path | Full path name on the client host |
| exe_user | User name on the execution host |
| exe_host | Execution host name |
| exe_path | Full path name on the execution host |

Please configure a staging script so that it can perform actual file transferring according to the content in the line corresponding to the staging number in Staging Order File.

[Example of Stage Order File]

```
0000 0000 cuser0 chost0 /home/user0/dir0/ euser0 ehost0/var/opt/nec/nqsv/jsv/jobfile/0.1508.105/stgfile/dir0/
0000 0001 cuser0 chost1 /home/user1/file1 euser0 ehost1/var/opt/nec/nqsv/jsv/jobfile/0.1508.105/stgfile/file1
```

In case the space character or the backslash is included in cli_path or exe_path, it will be converted into the character string of the table below in the stage order file.
Please note that in creating a staging script.

| Original character | Converted character |
|---|---|
| Space (' ') | "\s" |
| Horizontal tab ('\t') | "\t" |
| LF ('\n') | "\n" |
| CR ('\r') | "\r" |
| backslash ('\\') | "\\" |

[Example of the Staging Script]

```
#!/bin/sh
#
# User-defined Staging Script.(scp version)
#

while read stgno jobno cli_user cli_host cli_path exe_user exe_host exe_path
do
  [ ${stgno} != ${STG_STGNO} ] && continue

  src=""
  dst=""
  case ${STG_DIRECTION} in
  STAGE_IN)
    src=${cli_user} @${cli_host}:${cli_path}
    dst=${exe_user} @${exe_host}:${exe_path}
    ;;
  STAGE_OUT)
    src=${exe_user} @${exe_host}:${exe_path}
    dst=${cli_user} @${cli_host}:${cli_path}
    ;;
  esac
  su ${STG_USER} -c "scp -rp ${src} ${dst}" 2>& 3 || exit 1

done < ${STG_ORDERFILE}

exit 0
```

- Output of Staging Script

The staging script opens the standard output, the standard error output, and the 3rd file descriptor when starting. (Other file descriptors are closed.) The output destination of each file descriptor is as follows.

- The standard output, the standard error output

  They are redirected to the log file on the batch server if the log level is set two or more. When the length of message exceeds 1023 bytes (the maximum length of one line) it is divided into two or more lines.

- 3rd file descriptor

  The last line of message output to the 3rd file descriptor is forwarded to the batch server as an error message of transferring failure. This message can be displayed by qstat command with **-T, -f** option. The part that exceeded 127 bytes is deleted.

Example of outputting the batch server log is as follows.

```
06/29 12:01:00 NQSV (STAGE): nqs_stgd (netque0): host00: No address associated
with hostname
```

- Exit Code

The Staging Server judges the result of transferring by the exit code of the Staging Script and performs as follows.

| End code | Meaning | Perform |
|---|---|---|
| 0 | Success | [STAGE_IN]<br>The request transits to QUEUED state as the Stage-in success.<br>[STAGE_OUT]<br>The request transits to EXITED state as the Stage-out success. |
| 1 | Error but it can be retried | [STAGE_IN]<br>The request stays in the STAGING state, and the Stage-in is retried after the regular intervals of time.(*1)<br>[STAGE_OUT]<br>The request stays in the EXITING state, and retries the Stage-out again after the regular intervals of time.(*1) |
| 2 or more | Fatal error | [STAGE_IN]<br>The request returns to the QUEUED state as Stage-in failure and retries from the state of STAGING again after the regular intervals of time. |

| | | |
|---|---|---|
| | | [STAGE_OUT]<br>The request transits to the EXITED state as Stage-out failure. In this case, the Stage-out files are not forwarded. |

(*1) The transmission waiting time set to a network queue (please refer to 2.3.5. Routing Retry Interval.)

[Notes]

When an error that can be retried occurs in case of the external staging, the network queue that has the error stalls until retry. Therefore other network requests in the network queue also wait for it.

### 5.1.4. Job Migration

The Job Migration is a function to move batch jobs controlled by a job server to under control of the other job server. It also moves optional files user designated as well.

(1) Execution procedure

Execute job migration in the NQSV as follows.

1. Hold the target job

   Check Job Status.

```
$ qstat -J 123.bsv0.example.com
JNO RequestID        EJID   Memory      CPU JSVNO  ExecutionHost   UserName Exit
---- ---------------- ----- -------- -------- ----- ---------------- -------- ----
   0 123.bsv0.exampl   136   10.15G   184.24     6 ehost1           user        -
   1 123.bsv0.exampl    12    0.00B     0.00     7 ehost2           user        -
```

   Hold the whole request.

```
$ qhold 123.bsv0.example.com
Hold request 123.bsv0.example.com is accepted.
```

   Check if the parent request is in HELD state.

```
$ qstat 123.bsv0.example.com
RequestID       ReqName  UserName Queue     Pri STT S   Memory      CPU   Elapse R H M Jobs
--------------- -------- -------- -------- ---- --- - -------- -------- -------- - - - ----
123.bsv.example test1    user     execque1    0 HLD -    0.00B     0.00       68 Y Y Y    2
```

2. Execute the "migrate job" sub-command of qmgr(1M) command so that job database and migration file can be

   Migrate the batch job with the job ID 0:123.bsv0.example.com to the job server with job server number 8.

```
$ qmgr -Po
Mgr: migrate job = 0:123.bsv0.example.com job_server_id=8
Migrated Job
```

   Check if the job ID 0:123.bsv0.example.com is existent on the job server of job server

number 8.

```
$ qstat -J 123.bsv0.example.com
JNO  RequestID        EJID   Memory      CPU JSVNO ExecutionHost   UserName Exit
---- ---------------- ----- -------- -------- ----- ---------------- -------- ----
   0 123.bsv0.exampl     -    0.00B     0.00     8 ehost3           user        -
   1 123.bsv0.exampl     -    0.00B     0.00     7 ehost2           user        -
```

3. Release the target request

```
$ qrls 123.bsv0.example.com
Request 123.bsv0.example.com was release.
```

Execution is started again.

```
$ qstat 123.bsv0.example.com
RequestID       ReqName  UserName Queue     Pri STT S   Memory      CPU   Elapse R H M Jobs
--------------- -------- -------- -------- ---- --- - -------- -------- -------- - - - ----
123.bsv.example test1    user     execque1    0 RUN -   20.31G  1311.20      161 Y Y Y    2
```

(2) Target File for Migration

The job migration process is performed by transferring necessary files for controlling batch job execution to the target job server. The target file for this file transfer is as follows.

| File type | Path | Note |
|---|---|---|
| Job database | /var/opt/nec/nqsv/jsv/<jsvno>/<jobid>/ | Job management information |
| Job file | /var/opt/nec/nqsv/jsv/jobfile/<jobid>/ | Restart file, staging file, etc. |
| Migration file | Optional path | Checkpoint files, user specified files |

A job database and a job file are forwarded by NQSV to the time of a job Migration automatically. It is possible to forward the file a program is using as migration file or a necessary file for transmission by using -G option of qsub(1).

The file types that can be transferred are normal file, directory, and FIFO (named pipe). Other types are ignored. However, a symbolic link file is transferred as a normal file.

The target file that does not exist in the source is ignored. Moreover, if the file of the same name already exists on the destination, the target file will not be transferred. The transferred files along with the job migration will be automatically deleted by NQSV at the timing that the target job is completed or deleted.

(3) Transmission parameter

To optimize a file transfer, it is possible to set parameters for each job server. The parameters are as follows.

- Host name of the network interface
- Socket buffer size
- I/O buffer size

Execute the "set job_server migration_file_transfer_parameter" sub-command of qmgr (1M).

Example of outputting the batch server log is as follows.

```
$ qmgr -Po
Mgr: set job_server migration_file_transfer_parameter interface_hostname = host1
socketbuffer_size = (0) iobuffer_size = (1048576) job_server_id = 1
Set Migration_file_transfer_parameter: Job_Server_ID (1)
```

(4) Sharing Job Server Database

For migration, a file, such as job database or job files, is usually transferred between the job server hosts. But when the job server's database (/var/opt/nec/nqsv/jsv) is set for sharing area, is it possible to share the file on the sharing file system.

However, please also note that the file I/O on the sharing file system is slower in general than on the local file system.

> [Notes]
> Use NFS, ScaTeFS for the shared file system. Operations cannot be guaranteed if other shared file system is used.

Execute the following procedures to share the job server database:

1) Stop operations for all job servers executing on the execution host.
2) Set up the file system on the NFS server so that each execution server can access there through NFS mounting. As the job server will access the database using root privilege, please enable root accessing.
3) Please mount above file system to each execution host with NFS-mount.
   Specify /var/opt/nec/nqsv/jsv as the mount point and allow reading and writing in mounting the NFS file system.
4) Reboot the job servers.

### 5.1.5. Cleaning up of submit failed connected request

This function removes the remaining connected request automatically, if it failed to submit connection request.

Case disable this function, if the connection request failed to submit, a part of the connected request that already submitted is kept in HELD state.

The use of cleaning up of submit failed connected request function can be changed by the following qmgr sub-command.

```
set batch_server auto_delete_failed_request { on | off }
```

on Enable cleaning up of submit failed connected request function (default)
If the connection request failed to submit, the remaining connected request

is removed automatically.

off Disable cleaning up of submit failed connected request function

If the connection request failed to submit, the remaining connected request is not removed. It is maintained in the HELD state.

This operation needs manager privilege.

It is possible to refer to the above settings set by using qstat -Bf (batch server information).

```
$qstat -Bf
Batch Server: bsv.example.com
     :
   Allow Absolute Exepath = OFF
   Auto Delete Failed Request = ON
     :
```

### 5.1.6. Limit the number of re-runs

Sometimes a node running a batch request encounters a failure and is unable to continue running the request job. In such cases, NQSV may rerun the request and reexecute it. However, if the job is the cause of a node failure, unlimited reruns of the request may result in all nodes going into a failure state. This situation can be avoided by setting a limit on the number of re-runs for batch requests. The batch request rerun limit is done on a per-queue basis using a subcommand of the qmgr(1M) command.

| Queue type | Subcommands of qmgr(1M) |
|---|---|
| batch queue | set execution_queue rerun_default = {yes \| no} [limit = <limit>] <queue-name> |

```
$ qmgr -Pm
Mgr: set execution_queue rerun_default = yes limit = 1 batch1
Set rerun_default(limit:1). queue: batch1
```

If yes is specified, the batch request can be rerun. Set <limit> to the number of times it can be rerun. If not specified, or if 0 is specified, the number of re-runs is unlimited. The range of possible values is 0 to 2147483647; if no is specified, the batch request cannot be rerun.

If the number of re-runs for a batch request reaches the limit, the request will be put into the re-run disabled state (same as qsub -r n). If the situation becomes one in which re-runs occur again, the request will be deleted.

The re-run count limit applies to all cases that are re-run by NQSV.

- When a job is stalled and rerun by LINKDOWN of a job server
- When a job is re-run by the node health check function
- When a job is re-run by qrerun(1) command
- When a job is re-run by the forced re-run function of JobManipulator
- When a lower urgency request is rerun by assigning an urgent request
- When a request that has jobs is moved to the another queue by qmove(1) command with -f option

### 5.1.7. Automatic rerun and billing exclusion function for HW failures

If a node executing batch requests encounters a failure and the job server LINKDOWN, the request will be in a stalled state. If the stalled request exceeds the elapsed time (elaps) and ends, we delete the request and bill for the successfully completed job.

However, if there are abnormalities in the execution results due to the failure of the encountered job, even if there are jobs that have ended normally, the results may be unreliable. By enabling the automatic rerun and billing exclusion function for HW failures, we can rerun requests that exceed the elapsed time in a stalled state and exclude them from being billed.

This function can be enabled by configuring the nqsd.conf (/etc/opt/nec/nqsv/nqsd.conf) file and asvd.conf (/etc/opt/nec/nqsv/asvd.conf) file with the following format, restarting the batch server or executing the qmgr(1M) load nqsd_conf subcommand, and restarting the accounting server.

nqsd.conf

```
    enable_jsv_failure_handling on
```

The default value is off.

asvd.conf

```
    NO_CHANGE_ON_HW_FAILURE=on
```

The default value is off.

If you have billing enabled and want to exclude requests encountered due to HW failures caused by LINKDOWN from being billed, please enable the settings on both the batch server (BSV) and accounting server (ASV).

## 5.2. Interactive request

### 5.2.1. Request attribute modification

It is possible to change the attribute set to interactive request by the qalter (1) command like a batch request. Please refer to the qalter (1) command about the changeable attributes.

### 5.2.2. User EXIT

It is possible to set the user EXIT like a batch request to interactive request. About a setting method, please refer to 5.1.2.User EXIT.

### 5.2.3. Waiting Option

At the time of investment of interactive request by qlogin(1), if there are enough execution hosts to be assigned immediately, it is possible to select whether wait for enough execution hosts or cancel the request.

About a way of setting, please refer to 4.2.2(10) Waiting Option.

### 5.2.4. Compulsion Execution Shell

A login shell is usually used as a login shell for interactive queue session. However, when a compulsion execution shell is set as interactive queue, the shell is used as a login shell for interactive queue session.

About a setting, please refer to 4.2.2(11) Compulsion Execution Shell.

### 5.2.5. Idle Timer

A login shell for interactive queue session terminates after waiting for a certain time set as idle timer if input does not arrive. It is possible to set an idle timer every interactive queue.

About a setting, please refer to 4.2.2(12) Idle Timer.

### 5.2.6. Notes

- It is not possible to do file staging and a job migration on interactive request.
- Interactive request could not be submitted to routing queue.
- Firewall on the client host must be configured to use interactive request. Because it connects from execution host to client host that qlogin/qrsh command is executed. Please refer to Part 1.6.3 Configuration for interactive request.

# 6. Client's Management

A NQSV client is a general term for user commands, which are submission/reference/operation commands and administrator's commands.

## 6.1. Setting of api_client.conf

All NQSV client commands connect the batch server. It is possible to designate the batch server to an option of command, but when being not designated to command option, the batch is acquired from /etc/opt/nec/nqsv/api_client.conf file.
The syntax of api_client.conf is composed of a keyword and its value in a pair.

- The portion after **#** until the line end is regarded as a comment.
- The following keywords can be specified. The keywords are case-insensitive.

| Keyword | Description |
|---|---|
| batch_server_host | Specifies the host name of a batch server host. |
| batch_server_port | Specifies a TCP port number for connection to the batch server. |

[Sample of api_client.conf]

```
#
# NQSV Client Configuration file.
#

batch_server_host bsv0.example.com
batch_server_port 602
```

In case it fails to connect the batch server specified in the line of " batch_server_host", it is possible to register an another batch server host in the file as an alternative batch server to automatically connect to. The alternative batch server is specified under the line of " batch_server_host" by using the keyword, "alt_server_host".

```
#
# NQSV Client Configuration file.
#

batch_server_host   bsv0.example.com
alt_server_host     bsv1.example.com
batch_server_port   602
```

According to this setting, the client commands tries to connect the batch server specified in the line of "batch_server_host" first. In case they fail to connect, they connect the host specified in the line of "alt_server_host". And, if it fails, too, the commands return error for connection failure of batch server. When the client commands connect the host specified in the "alt_server_host" after they fail to connect the batch server specified in the line of

"batch_server_host", they display name of the batch server host to the standard error output.

```
$ qsub job_script
Connected to bsv1.example.com.
Request 204.bsv1.example.com submitted to queue: batch.
```

But, in case that the batch sever to connect is explicitly specified by qmgr command or the client commands, alternative server host is not automatically connected even if an alternative server is specified in the line of "alt_server_host".

> [Notes]
> It is necessary to thoroughly check whether the request specified as a target of operation is really the target request when processing a request, because an another batch server is automatically connected according to the "alt_server_host" setting in case it fails to connect the default batch server.

### 6.2. User Agent

It is necessary to start the user agent to return the result file or do staging of an input/output file with the client host.

The systemctl is usually used to start or end the user agent.
The user agent has to be specified the batch server that permits a connection as an argument at the time of a start, and set the host name of the batch server to the BSV_HOSTS variable in /etc/opt/nec/nqsv/nqs_uag.env file. (For more than one, punctuate by space.)

```
BSV_HOSTS="bsv1.example.com bsv2.example.com bsv3.example.com"
```

To start user agent automatically when client host is booted, execute following command

```
# systemctl enable nqs-uag.service
```

<u>Activation</u>

If nqs-uag service is enabled, starts as follows by a root privilege using the systemctl.

```
# systemctl start nqs-cui.target
```

If nqs-uag service is not enabled, execute following command to start nqs-uag.service.

```
# systemctl start nqs-uag.service
```

For manual activation, specify the host name of the batch server that permits a connection as an argument as follows.

```
# /opt/nec/nqsv/sbin/nqs_uagd bsv0.example.com bsv1.example.com
```

When the user agent is activated, it binds TCP Port 603 by default and waits connection requests from the batch server (from the staging server). A port number specified in /etc/services with the name "nqsv-uag" or "nqsII-uag" will be given priority. If both of "nqsv-uag" and "nqsII-uag" specified, "nqsv-uag" will be given priority.

<u>Termination</u>

If nqs-uag service is enabled, terminates as follows by a root privilege using the systemctl command.

```
# systemctl stop nqs-cui.target
```

If nqs-uag service is not enabled, execute following command to stop nqs-uag.service.

```
# systemctl stop nqs-uag.service
```

Send SIGTERM to the appropriate process when terminating the user agent manually.

### 6.3. Configuration for interactive request and attach to request

On the interactive request, execution host that executes the job makes TCP connection to the client host that execute qlogin/qrsh command. This connection uses ephemeral port of client host. Set the firewall on the client host to accept the connection for ephemeral port from execution host.

It also need same configuration as interactive request to attach to the request.

# 7. Remote Execution by Interactive Function

The remote execution by interactive function is that the user does a program execution request on the client host, and the interactive request executed on a remote host is put in NQSV automatically and a program is executed on the execution host.

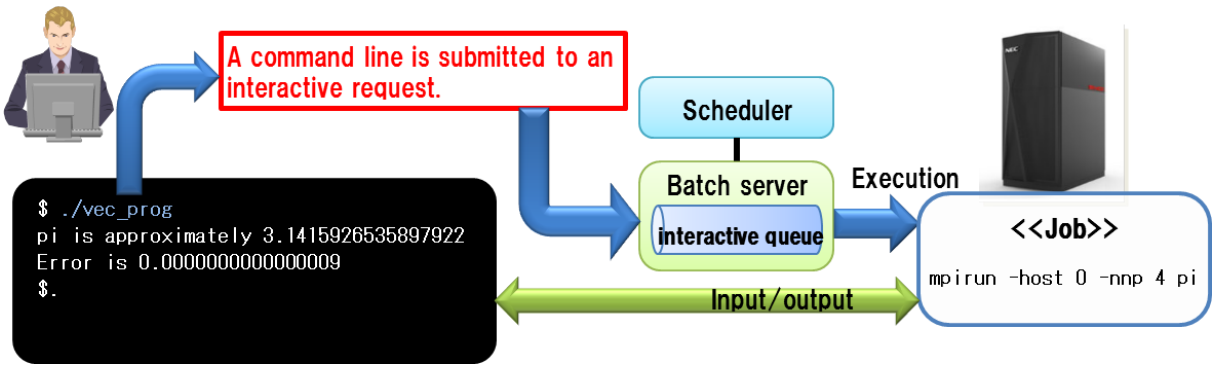



Figure 7-1 : Remote execution by interactive request

The qrsh command puts interactive request in NQSV. It is necessary to register a command line of the program and options for qrsh in advance. In addition, it is necessary to make remote execution program to submit an interactive request by using qrsh. The qcmdconf (1) command is used for this.

## 7.1. Register

To register the program to be executed remote host, -a option of qcmdconf(1) is used.

```
qcmdconf -a --name=entry-name --cmd=remote_cmd-line
            --queue = queue-name [--opt=qrsh_options] [-f file-name] [-S]
```

--name=*entry-name*

Specify the name of the registration name of the command line to be executed on a remote host. When not specified the following-f option, this name will be the file name of the remote execution program.

--cmd = *remote_cmd - line*

Specify the command line to be executed on a remote host. The number of characters of the command line is at most 1023 characters.

If there are SPACE for more than one in the command line, surround with single quote (') or double quote ("). When there are control characters, put escape character (\).

--queue = *queue-name*

Specify the interactive queue name.

--opt=*qrsh_options*

Specify options, such as restriction of resources.

If there are SPACE more than one, surround with single quote (') or double quote (").

-f *file-name*

Specify the file name of the remote execution program.

-S

It is registered by the administrator privilege. It is only the root user that can specify this option. The remote execution program available is registered for all users.

Without -S options, the remote execution program is for the user's individuals who executed the qcmdconf command. (A registration data file is made in the user's home directory.)

Registered data are stored in the following location.

- For the general user:
  $ HOME/.nqsv/qcmd_list  Registration data for the user's individuals
- For administrator right designation (-S option specified by the root privilege):
  /etc/opt/nec/nqsv/qcmd_list  Registration data of system

Below is an example.

```
$ qcmdconf -a --name vecprog --cmd='mpirun -host 0 -nnp 4 -ve 0 ./pi' --queue iq1 --
opt='-T necmpi --cpunum-lhost=4 --venum-lhost=1'
$ ls -l vecprog
-rwxr-xr-x 1 user1 grp1 37  Feb  14 14:54 2017 vecprog.
```

The remote execution program "vecprog" was made in a current directory.

When this "vecprog" is started, an interactive request is put in the interactive queue iq1 with "-T necmpi --cpunum-lhost=4" for the qrsh command.

The following command line will be executed on the execution host in this case.

```
mpirun -host 0 -nnp 4 -ve 0 ./pi
```

***** Host specified to mpirun

The job number, not the host name, is specified as an argument of -host option to the interactive request like a batch request.

### 7.2. Reference to the Information

It is possible to refer to registration information on the remote execution program by -l option

of qcmdconf (1).

```
qcmdconf -l [-S]
```

A list of registration information on the remote execution program that can be used is shown to standard output from the information on system and user's individuals.

-S

Show the information registered by the administrator privilege. When executed by the administrator privilege, list of the remote execution type program registration information is shown only from registration information on system.

[Use example for general users]

```
$ qcmdconf -l
------------------------------
Common configuration
------------------------------
Name         remote-cmd                    queue    qrsh-option
--------------------------------------------------------------------------------
AP1          "AP1 -x"                      iqs      "--cpunum-lhost=2"
------------------------------
User configuration
--------------------------------------------------------------------------------
name         remote-cmd                    queue    qrsh-option
--------------------------------------------------------------------------------
vecprog      "mpirun -host 0 -nnp 4 /opt/nec/ve/bin/ve_exec ./pi" iq1 "-T necmpi --
cpunum-lhost=4 --venum-lhost=1"
```

[Use example for the administrator privilege]

```
# qcmdconf -l -S
------------------------------
Common configuration
------------------------------
name        remote-cmd                    queue    qrsh-option
--------------------------------------------------------------------------------
AP1         "AP1 -x"                      iqs      "--cpunum-lhost=4"
```

### 7.3. Execution

When the remote execution program is submitted on the client host, according to the registration information (*), an interactive request will be automatically submitted by qrsh(1) and be executed on the execution host immediately.

* Refer to registration data $HOME/.nqsv/qcmd_list for the individual user's remote execution program.

At this time, the standard input/output error output of an executed program on the execution

host is connected with a client host.

[Example of the remote execution program]

```
$ ./vecprog
pi is approximately 3.1415926535897922, Error is 0.0000000000000009
```

By the above example, the following command line is executed on the execution host, and the standard output on the execution host is output in standard output of the client host.

```
    mpirun -host 0 -nnp 4 /opt/nec/ve/bin/ve_exec ./pi
```

Further, it is possible to redirect the standard output/error output on the execution host by using a redirection or a pipe.

```
$ ./vecprog > outfile
```

A "outfile" is made on the client host (the user's execution terminal).

It is possible to give arguments to the remote execution program. When arguments are specified as the time of starting remote execution program, they are passed to the execution command line on the execution host just as it is.
For example, when an argument is specified by a command line on a client host:

```
$ ./vecprog -opt1
```

The following command line is executed on the execution host.

```
    mpirun -host 0 -nnp 4 /opt/nec/ve/bin/ve_exec ./pi -opt1
```

As the remote execution program is automatically submitted by interactive request function, the information is shown as that by qstat(1).

```
$ qstat
RequestID       ReqName  UserName Queue     Pri STT S   Memory      CPU   Elapse R H M
Jobs
--------------- -------- -------- -------- ---- --- - -------- -------- -------- - - - ---
-
17608.bsv1      QRSH     user1    iq1         0 RUN -    0.00B     0.00        2 N N N
1
```

### 7.4. Deletion

To delete the program to be executed remote host, -d option of qcmdconf(1) is used.

```
qcmdconf -d entry-name [-S].
```

A registration name for the remote execution program is specified.
Only registration information will be deleted. To delete the remote execution program file, use rm command.

*entry - name*

Specify registration name of the remote execution program to be deleted.

-S

Delete the information registered by the administrator privilege.

Without -S options, the remote execution program for the user's individuals will be deleted.

### 7.5. Common Remote Execution Program

When doing registration and making of the remote execution program by qcmdconf -a by the administrator privilege (With -S option), the registration information are stored at /etc/nsqII/qcmd_list and the remote execution program will be available for all users.

The location of the registration data is fixed and it is /etc/opt/nec/nqsv/qcmd_list, but the location of remote execution program made by the information is not fixed.

When doing the practical use by using the remote execution program in common shared by several users as tool, the execution program needs to be stored the location which can be referred to the users, and set the path to each user environment.

When the root user registers without -S option, the remote execution program can be used by root user only. Other users cannot execute the remote execution program.

# 8. Preservation of NQSV Environment

## 8.1. Environment of Batch Server and Queue

To preserve the information about a batch server, execution hosts (job servers), node groups and queues, use the "list all" sub-command of qmgr(1M).

```
# qmgr
Mgr: list all file=savefile.bsvque
```

The information is changed to the format of the sub-command of qmgr and preserved in a file (savefile.bsvque). When template information or custom resource information are defined, it is preserved too.

It is possible to restore the NQSV environment easily by qmgr command.

```
# qmgr -Pm < savefile.bsvque
Set Logfile Path.
Set Log Level.
Set Save Logfile Number.
Set Logfile Size.
Set Heartbeat Interval.
Set Load Interval.
      :
```

Information about the bind relation of execution queue is not outputted, because there is a possibility that the change of formation about job servers and scheduler. To output the information, use the "list bind" sub-command.

To preserve only template information or only custom resource information, use "list template" or "list custom_resource" sub-command.

## 8.2. Environment of Batch Server

To preserve the information about a batch server, use the "list batch_server" sub-command of qmgr(1M).

```
# qmgr
Mgr: list batch_server file=savefile.bsv
```

The information is changed to the format of the sub-command of qmgr and preserved in a file.

It is possible to restore the NQSV environment easily by qmgr command.

```
# qmgr -Pm < savefile.bsv
Set Logfile Path.
Set Log Level.
Set Save Logfile Number.
Set Logfile Size.
Set Heartbeat Interval.
Set Load Interval.
      :
```

To preserve information about execution hosts (job servers) and node groups, user the "list node" sub-command of qmgr(1M).

### 8.3. Environment of Queue

To preserve the information about queues, use the "list queue" sub-command of qmgr(1M).

```
# qmgr
Mgr: list queue file=savefile.queue
```

The information is changed to the format of the sub-command of qmgr and preserved in a file. It is possible to restore the NQSV environment easily by qmgr command.

```
# qmgr -Pm < savefile.queue
Queue exec1 created.
Enable Queue: exec1
Start Queue: exec1
Set Checkpoint: exec1
Set Access_List Noaccess.
Set Reserve ID: exec1
Set Restart option: exec1
Set Elapse Time (Per-Request) limit: exec1
Set standard Elapse Time (Per-Request): exec1
Set CPU Time (Per-Job) limit: exec1
Set standard CPU Time (Per-Job): exec1
      :
```

### 8.4. Binding

To preserve the information about binding between queue and job server, use the "list bind" sub-command of qmgr(1M).

```
# qmgr
Mgr: list bind file=savefile.bind
```

The information is changed to the format of the sub-command of qmgr and preserved in a file. It is possible to restore the NQSV environment easily by qmgr command.

```
# qmgr -Pm < savefile.bind
Bound Job_Server_ID (0) to Queue (exec1).
Bound Job_Server_ID (1) to Queue (exec1).
Bound Job_Server_ID (2) to Queue (exec1).
      :
```

# 9. MPI Request Execution Environment Settings

In this section, the following MPI execution environment settings are explained.

- OpenMPI (OpenMPI Version 1.8.3, Version 2.0.1, Version 2.1.2, Version 3.0.0, Version 4.0.3, Version 4.1.1, Version 4.1.3)
- IntelMPI (Intel(R) MPI Library for Linux OS Version 4.1 Update 3, Version 5.1 Update 3, 2017 Update 1, 2018 Update 3, 2019 Update 7, 2021 Update 5)
- Intel OneAPI (Intel(R) OneAPI Toolkits Release 2021.4)
- MVAPICH2 (MVAPICH2 Version 2.0, Version 2.3.4, Version 2.3.6, Version 2.3.7)
- Platform MPI (Platform MPI Version 09.01.04.03)

## 9.1. OpenMPI Environment Settings

(1) Execution Host Settings

To execute an OpenMPI job, set installation path of OpenMPI in the script file (/opt/nec/nqsv/sbin/startopenmpi.sh) for starting OpenMPI job on each execution host.

**startopenmpi.sh**

```
    :
# Installation path of your OpenMPI
OMPIHOME=/opt/openmpi-1.8.3
    :
```

The startopenmpi.sh script sets PATH and LD_LIBRARY_PATH and passes them to the slave job. If you want to execute a job that depends on the path setting order in your environment, set PATH and LD_LIBRARY_PATH appropriately according to the environment.

(2) Job and MPI Process Allocation

NQSV starts OpenMPI process in each job cooperating with the remote process generation function of OpenMPI.

1. NQSV creates as many jobs as specified by the command to submit a request and provides MPI with the name of host allocated for jobs and the number of CPUs.
2. MPI allocates MPI process based on the information provided by NQSV.
   MPI is based upon the premise that a job is usually allocated per a node.

[Notes]
OpenMPI program execution fails if more than on jobs are allocated on an execution host. A job needs to be allocated on each execution host.

### 9.2. IntelMPI Environment Settings

NQSV starts IntelMPI process in each job cooperating with the remote process generation function of IntelMPI.

1. NQSV creates as many jobs as specified by the command to submit a request and provides MPI with the name of host allocated for jobs and the number of CPUs.
2. MPI allocates MPI process based on the information provided by NQSV.
   MPI is based upon the premise that a job is usually allocated per a node.

[Notes]
If multiple jobs are allocated on an execution host, IntelMPI processes are disproportionately allocated in only one of those jobs and the job cannot be executed normally.
A job needs to be allocated on each execution host.

### 9.3. MVAPICH2 Environment Settings

NQSV starts MVAPICH process in each job cooperating with the remote process generation function of MVAPICH2.

1. NQSV creates as many jobs as specified by the command to submit a request and provides MPI with the name of host allocated for jobs and the number of CPUs.
2. MPI allocates MPI process based on the information provided by NQSV.
   MPI is based upon the premise that a job is usually allocated per a node.

[Notes]
MVAPICH program execution fails if more than one job is allocated on an execution host.
Only one job needs to be allocated on each execution host.

### 9.4. PlatformMPI Environment Settings

NQSV starts PlatformlMPI process in each job cooperating with the remote process generation function of PlatformMPI.

1. NQSV creates as many jobs as specified by the command to submit a request and provides MPI with the name of host allocated for jobs and the number of CPUs.
2. MPI allocates MPI process based on the information provided by NQSV. MPI is based upon the premise that a job is usually allocated per a node.

[Notes]

If multiple jobs are allocated on an execution host, PlatformMPI processes are disproportionately allocated in only one of those jobs and the job cannot be executed normally.

A job needs to be allocated on each execution host.

### 9.5. NEC MPI Environment Settings

NQSV starts NEC MPI process in each job cooperating with the remote process generation function of NEC MPI.

It is possible to configure the queue to be submitted whether to use Hydra or MPD as the process manager for executing NEC MPI jobs. To configure the process manager for NEC MPI jobs for a queue, run qmgr(1M) with operator privileges and use the following subcommand.

```
set execution_queue necmpi_process_manager = { hydra | mpd } queue
```

You set the process manager to be used by NEC MPI requests submitted to the queue specified by *queue* to hydra or mpd. The default process manager when the queue is created is mpd.

> This setting is only available for batch queues. Interactive queues always use hydra as the process manager.

The process manager of NEC MPI set in the queue can be checked by qstat -Qf as follows.

```
$ qstat -Qf bq
Execution Queue: bq@bsvhost
    Run State       = Active
    Submit State = Enable
    Scheduler ID     = 1
        :
    GPU-CPU Affinity = OFF
    GPU-CPU Affinity CPU per GPU = 1
    NEC MPI Process Manager = mpd
    Range of Jobs Limit per Batch Request (min,max) = 1,10240
    Range of Request Priority = {
        :
```

## 10. Group of Request

User and group information are retained in the request information. This group usually

becomes primary group of user who submits a request, and a job is executed by the primary group on execution host.

If designated group execution function for request is used, the designated group becomes the group of request by contrast, and a job is executed by the designated group. In this section, the designated group execution function is explained.

## 10.1. Designated Group Execution Function for Request

This is the function to execute a job and perform accounting by the designated group (or supplementary group) when submitting a request.

It is possible to designate group (supplementary group) by --group=<group_name> option of the request-submitting command (such as qsub, qlogin and qrsh) as the group of the request. In case --group option is not specified, the group of the request-submitting command becomes the group of request. In addition, it is possible to designate supplementary group by switching execution group to supplementary group by newgrp and by executing the request- submitting command.

When group of a request is designated by using this function, the followings are to be done for the request.

- Queue access limitation, submit number limit and queue resource restriction are to be checked by the designated group for the request.
- A job is to be executed by the designated group as execution GID.
  - If the designated group does not exist on the execution host, the job shall fail.
- The designated group for the request is to be registered as group information of request account and job account.
- A file is to be created by the designated group in case of internal staging during the file staging for the request.
- In case of external staging, the value of the environment variables, STG_GROUP (group name) and STG_GID (group GID), set to staging script are the group designated by the user.

## 10.2. Enable/Disable Function and Reference of Settings

Group designated execution function for request is enabled by using the following the sub-command of qmgr.

Format

```
set batch_server specify_group_request { on | off }
```

on　　Enable group designated execution function for request
Supplemental group can be specified on submitting request and a job is executed by the group.

off　　Disable group designated execution function for request

This operation needs operator privilege.

It is possible to refer to the above settings set by using qstat -Bf　(batch server information).

```
$qstat -Bf
Batch Server: bsv.example.com
      :
   Specify Group for Request = ON
   Allow Absolute Exepath = OFF
      :
```

### 10.3. Usage Precautions

Note as follows regarding usage of this function.

- Among NQSV systems such as client host, BSV host and job server host, setting of user (UID) and group (GID) need to be in common. If they are not in common, submit error, execution error, or job execution by unintended UID/GID may occur.
- Do not use user mapping.
  In case that user mapping is used (user is mapped to a different user in any of client host, BSV host or job server host), the designated group on submitting request is to be invalid and request is to be operated by the primary group of the mapped user.
- The following settings need to be in common among BSV hosts and in case of transferring request to other BSV via routing queue.
  - Enable/Disable of this function
  - Settings of user (UID) and group (GID)
  - Settings of user mapping
- Regardless of access enable or disable check state for the access limitation of the queue, access limitation is to to checked against the group designated on submitting request.
- Group cannot be designated for workflow (wstart). However, it is possible to designate group for individual request in workflow.

# 11. Limit per Group and User

Regarding limit for batch server and queues, some upper limit can be set by individually specifying group or user name.
Targets of limit are submit number limit for batch server, batch queues, interactive queues and routing queues and limitation of job number to create and elapse time limit for batch queue and interactive queue.

## 11.1. Submit Number Limit per Batch Server

For batch server, upper limit of submit number of request can be set by individually specifying group or user name.
In case of specifying group name for submit number limit, submit number limit per a group is invalid and in case of specifying user name, submit number of limit per a user is invalid.

(1) Limit Settings

The sub-commands of qmgr (1M), "set batch_server" to set submit limit specifying group name or user name individually are as follows. The operator privilege or higher is necessary to set the limit.

| Attribute | qmgr(1M) Sub-Command |
|---|---|
| The number of requests that specified group can submit to a queue | set batch_server group_submit_limit=<limit> groups=<group_name> |
| The number of requests that specified user can submit to a queue | set batch_server user_submit_limit=<limit> users=<user_name> |

(2) Limit Cancellation

The sub-commands of qmgr (1M), "delete batch_server" to cancel submit limit specifying group name or user name individually are as follows. The operator privilege or higher is necessary to set the limit.

| Attribute | qmgr(1M) Sub-Command |
|---|---|
| The number of requests that specified group can submit to a queue | delete batch_server group_submit_limit groups=<group_name> |
| The number of requests that specified user can submit to a queue | delete batch_server user_submit_limit users=<user_name> |

## 11.2. Limitation per Queue

### 11.2.1. Submit Limit

It is possible to set a limit per queue to the number of requests that can be submitted to a batch queue, interactive queue and routing queue by individually specifying group name or user name.

In case of specifying group name for submit number limit, submit number limit per a group is invalid and in case of specifying user name, submit number of limit per a user is invalid.

However, it is not possible to exceed a submit number limitation value per queue.

(1) Limit Settings

The sub-commands of qmgr (1M), "set execution_queue", "set interactive_queue" and "set routing_queue" to set submit limit specifying group name or user name individually are as follows. The operator privilege or higher is necessary to set the limit.

| Attribute | QueueType | qmgr(1M) Sub-Command |
|---|---|---|
| The number of requests that specified group can submit to a queue | batch queue | set execution_queue group_submit_limit=<limit> groups=<group_name> <queue> |
| | interactive queue | set interactive_queue group_submit_limit=<limit> groups=<group_name> <queue> |
| | routing queue | set routing_queue group_submit_limit=<limit> groups=<group_name> <queue> |
| The number of requests that specified user can submit to a queue | batch queue | set execution_queue user_submit_limit=<limit> users=<user_name> <queue> |
| | interactive queue | set interactive_queue user_submit_limit=<limit> users=<user_name> <queue> |
| | routing queue | set routing_queue user_submit_limit=<limit> users=<user_name> <queue> |

(2) Limit Cancellation

The sub-commands of qmgr (1M), "delete execution_queue", "delete interactive_queue" and "delete routing_queue" to cancel submit limit specifying group name or user name individually are as follows.

The operator privilege or higher is necessary to cancel the limit.

| Attribute | QueueType | qmgr(1M) Sub-Command |
|---|---|---|
| The number of requests that specified group can submit to a queue | batch queue | delete execution_queue group_submit_limit groups=<group_name> <queue> |
| | interactive queue | delete interactive_queue group_submit_limit groups=<group_name> <queue> |
| | routing queue | delete routing_queue group_submit_limit groups=<group_name> <queue> |
| The number of requests that | batch queue | delete execution_queue user_submit_limit users=<user_name> <queue> |

| | | |
|---|---|---|
| specified user can submit to a queue | interactive queue | delete interactive_queue user_submit_limit users=<user_name> <queue> |
| | routing queue | delete routing_queue user_submit_limit users=<user_name> <queue> |

### 11.2.2. Limitation of the Job Number

It is possible to limit the number of jobs created from request submitted to batch queue or interactive queue specifying group name or user name. If job number set on submitting request is outside the range of job number set for group or user, submission to the queue shall be rejected.

However, even if this limit is set, it is not possible to exceed the number of jobs set to the queue.

(1) Limit Settings

The sub-commands of qmgr (1M), "set execution_queue" and "set interactive_queue" to set job number limit of a queue specifying group name or user name individually are as follows.

The operator privilege or higher is necessary to set the limit.

| Attribute | QueueType | qmgr(1M) Sub-Command |
|---|---|---|
| The range of number of jobs for requests that specified group can submit to a queue | batch queue | set execution_queue jobs_range = <limit> groups=<group_name> <queue> |
| | interactive queue | set interactive_queue jobs_range = <limit> groups=<group_name> <queue> |
| The range of number of jobs for requests that specified user can submit to a queue | batch queue | set execution_queue jobs_range = <limit> users=<user_name> <queue> |
| | interactive queue | set interactive_queue jobs_range = <limit> users=<user_name> <queue> |

(2) Limit Cancellation

The sub-commands of qmgr (1M), "delete execution_queue", "delete interactive_queue" to cancel job number limit of a queue specifying group name or user name individually are as follows.

The operator privilege or higher is necessary to set the limit.

| Attribute | QueueType | qmgr(1M) Sub-Command |
|---|---|---|
| The range of number of jobs for requests that specified group can submit to a queue | batch queue | delete execution_queue jobs_range groups=<group_name> <queue> |
| | interactive queue | delete interactive_queue jobs_range groups=<group_name> <queue> |
| The range of number of jobs for requests that specified user can submit to a queue | batch queue | delete execution_queue jobs_range users=<user_name> <queue> |
| | interactive queue | delete interactive_queue jobs_range users=<user_name> <queue> |

### 11.2.3. Elapse Time Limit

It is possible to limit elapse time for request submitted to batch queue or interactive queue specifying group name or user name. If elapse time set on submitting request is outside the range of elapse time set for group or user, submission to the queue shall be rejected.

However, even if this limit is set, it is not possible to exceed elapse time limit set to the queue.

In case that elapse time limit is not specified on submitting a request, minimal value of elapse time limit from the followings is set.

- Default value of elapse time limit for queue
- Maximum value of elapse time limit set for group
- Maximum value of elapse time limit set for user

(1) Limit Settings

The sub-commands of qmgr (1M), "set execution_queue" and "set interactive_queue" to set elapse time limit of a queue specifying group name or user name individually are as follows.

The operator privilege or higher is necessary to set the limit.

| Attribute | QueueType | qmgr(1M) Sub-Command |
|---|---|---|
| The elapse time limit for requests that specified group can submit to a queue | batch queue | set execution_queue per_req elapse_time_limit = <limit> groups=<group_name> <queue> |
| | interactive queue | set interactive_queue per_req elapse_time_limit = <limit> groups=<group_name> <queue> |
| The elapse time limit for requests that specified user can submit to a queue | batch queue | set execution_queue per_req elapse_time_limit = <limit> users=<user_name> <queue> |
| | interactive queue | set interactive_queue per_req elapse_time_limit = <limit> users=<user_name> <queue> |

(2) Limit Cancellation

The sub-commands of qmgr (1M), "delete execution_queue" and "delete interactive_queue" to cancel elapse time limit of a queue specifying group name or user name individually are as follows.

The operator privilege or higher is necessary to set the limit.

| Attribute | QueueType | qmgr(1M) Sub-Command |
|---|---|---|
| The elapse time limit for requests that specified group can submit to a queue | batch queue | delete execution_queue per_req elapse_time_limit groups=<group_name> <queue> |
| | interactive queue | delete interactive_queue per_req elapse_time_limit groups=<group_name> <queue> |
| The elapse time limit for requests that specified user can submit to a queue | batch queue | delete execution_queue per_req elapse_time_limit users=<user_name> <queue> |
| | interactive queue | delete interactive_queue per_req elapse_time_limit users=<user_name> <queue> |

### 11.3. Reference of Limit Information per Group and User

Check the information of each limit set specifying group and user individually by using qstat(1) command with --limit option.
Each settings of batch server can be checked with -Bf option and each settings of queues can be checked with -Qf option. But regarding limit set specifying group or user individually, only whether or not the limit is set can be checked with those options.

(1) `qstat --limit`

qstat --limit displays information about limitation set specifying group and user individually for batch server and queues such as batch queue, interactive queue and routing queue.
Displayed content differs according to access privilege on executing qstat(1) command as follows.

- Manager privilege and operator privilege
  Limit values for all of groups and users are displayed.
  - Submit number limit for batch server
  - Queue information
    - Access Restriction (user list and group list)
    - Submit number limit
    - Limitation of the job number
    - Elapse time limit
- Group Manager Privilege
  Limit values for the group managed by the group administrator are displayed.
  - Submit number limit for batch server
  - Queue information (only queues for which the administrator has access privilege)
    - Access Restriction (group list)
    - Submit number limit
    - Limitation of the job number
    - Elapse time limit
- Special User Privilege and General User Privilege
  Limit values for user and group who executes qstat(1) command are displayed.
  Queue information only for ones that access privilege is granted. Besides, list of access restrictions are not displayed.

Display example: qstat(1) --limit is executed with administrator privilege

```
$qstat -Pm --limit
Batch Server: bsvhost
  Submit Number Limitation Value       = 1000
```

```
  Submit User  Number Limitation Value = UNLIMITED
        user1       : Limit = 200
        user2       : Limit = 100
  Submit Group Number Limitation Value = 300
        grpA        : Limit = 100
        grpB        : Limit = 400

Execution Queue: exec1
  Access User list = {
      user2
  }
  Access Group list = {
      grpA     grpB     grpZ
  }
  Submit Number Limit       = 100
  Submit User  Number Limit = 10
        user1       : Limit = 30
        user2       : Limit = 5
  Submit Group Number Limit = 50
        grpA        : Limit = 40
        grpZ        : Limit = UNLIMITED
  Range of Jobs Limit per Batch Request (min,max) = 1,2048
      User  Limit:
            user1       : Limit = 1,512
            user2       : Limit = 1,256
      Group Limit:
            grpA        : Limit = 512,1024
            grpZ        : Limit = 512,2048
  Resources Limits:
    (Per-Req) Elapse Time Limit = Max: UNLIMITED Warn: UNLIMITED
      User  Limit:
            user1       : limit = Max:     3000S Warn:     2940S
            user2       : limit = Max:     1500S Warn:     1450S
      Group Limit:
            grpA        : limit = Max:      600S Warn:      540S
            grpZ        : limit = Max:     5000S Warn:     4900S

Execution Queue: bq1
    :
```

(2) qstat -Bf

In case that Submit number limit is set specifying group name or user name for batch server, the message, "The other Limitation Values are setting." is displayed.

Display example: qstat -Bf

```
$ qstat -Bf
Batch Server: bsvhost
       :
    Submit Number Limitation Value       = 1000
```

```
    Submit User  Number Limitation Value = UNLIMITED
    Submit Group Number Limitation Value = UNLIMITED
    The other Limitation Values are setting.
  Use License :
       :
```

(3) qstat -Qf

In case that either Submit number limit, Limitation of the job number, or elapse time limit is set specifying group name or user name for batch server, the message, "The other Limitation Values are setting." is displayed at the end of each queue information.

Display example: qstat -Qf

```
$ qstat -Qf exec1
Execution Queue: exec1@ bsvhost
       :
  Kernel Parameter:
    Resource Sharing Group     = 0
       :
    Aging Range                = 160
    Slave Priority             = 0
  The other Limitation Values are setting.

Execution Queue: exec2@ bsvhost
      :
```

# 12. VE and GPU Support

NQSV manages VE and GPU that is installed on the execution host and controls the job execution optimally on that hardware.

## 12.1. Configuration for VE environment

This configuration is required for the environment whose execution host is SX-Aurora TSUBASA system.
To execute the job which uses VE, CPU cores for VEOS must be reserved on VH. Therefore limit the CPU cores which is used for executing job by using CPUSET function to reserve the same number of CPU cores as the number of VEs on VH.

Configuration procedure in detail is explained below. For example the VH has 40 CPU cores and 8VE. Please refer to 18.2 CPUSET function together for details of the CPUSET function.

1. Create cpuset.conf file.

   Create cpuset.conf under /etc/opt/nec/nqsv of execution host.

2. Define the CPUSET of total resources on the host.

   Define the following configuration to cpuset.conf.

   ```
   cpuset   0-39    0-1    0
   ```

   This configuration indicates 40 cores (core number 0 to 39) and 2 memory nodes (sockets) are equipped on the host.

3. Define the CPUSET of CPU cores for executing job.

   Define the following configuration to cpuset.conf.

   ```
   forjob   8-39    0-1    1
   ```

   This configuration indicates 32 cores (core number 8 to 39) and 2 memory nodes (sockets) are reserved for job execution. It configuration is mapped to virtual RSG number 1. Remaining 8 cores (core number 0 to 7) are reserved for VEOS.
   Select the CPU cores in the socket that VE is connected as much for CPU cores for job execution

4. Start JSV.

   Start JSV process after above configuration.

5. Enable socket scheduling function of the queue.

   Enable socket scheduling function of the queue to use CPUSET.

Use following qmgr subcommand. (To configure to exec1 queue)

```
set execution_queue numa_control = on exec1
```

* There are few notes on the use of the socket scheduling feature. See Chapter 18 Socket Scheduling (18.1 Socket Scheduling function and 18.1.3 Memory allocation policy).

6. Assign the CPUSET number to the queue.

   Assign the CPUSET (virtual RSG) number 1 to the queue that VE job submit.

   Use following qmgr subcommand. (To configure to exec1 queue)

```
set execution_queue kernel_param rsg_number = 1 exec1
```

Job is executed by using 32 cores (core number 8 to 39) which defined on the forjob CPUSET and 8 cores can be reserved for VEOS by above configuration. Please configure the CPU core number appropriately according to the actual number of VE to be equipped on the execution host by making reference to this example.
In addition, if the Hyper-Threading feature is enabled, please calculate the number of logical cores. In the above example, 8 logical cores should be reserved for VEOS.

### 12.2. Submitting a request with the total number of VEs specified and Setting of the default number of incorporated VE nodes

This function is available only for the environment whose execution host is SX-Aurora TSUBASA system.
With the NQSV, the user can enter a request with the total number of VEs for the request specified. With this method, the number of jobs is automatically determined. Therefore, the user can easily enter requests for the number of VEs, without caring for the number of jobs.

When entering a request with the total number of VEs specified, specify the default number of incorporated VE nodes for the queue.
The default number of incorporated VE nodes means the default number of VE nodes incorporated in each execution host bound to the queue.
When the user enters a request with the total number of VEs specified, the batch server calculates the number of jobs based on the default number of incorporated VE nodes then applies it to the request.

Specify the default number of incorporated VE nodes by using the qmgr subcommands below, per execution queue. This setting requires administrator rights.

```
set execution_queue defined_ve_number = <venum> <queue_name>
```

```
set interactive_queue defined_ve_number = <venum> <queue_name>
```

For <venum>, specify an integer of 1 or larger. The default is 1.

You can check the specified default number of incorporated VE nodes as follows, by using the qstat -Qf command.

```
$ qstat -Qf bq
Execution Queue:bq@bsv1
    Run State    = Active
    Submit State = Enable
    ...
Defined VE Number     = 1
Submit VE Node Range (min,max)  = 0,UNLIMITED
Total Request         = 0
Arriving Request      = 0
    ...
```

Total number of VEs is specified by "qsub --venode" at submission of a request.
When the total number of VEs is specified, the number of jobs is calculated as below. Cut off after the decimal point.

```
<No. of jobs> = (<venum> + (<default no. of incorporated VE nodes> - 1)) / <default no. of incorporated VE nodes>
```

For example, if the default number of incorporated VE nodes for a queue is 8, the number of jobs are calculated as below, according to the value specified for the --venode option.

1. In case of qsub --venode=16, the number of job is 2
2. In case of qsub --venode=9, the number of job is 2
3. In case of qsub --venode=8, the number of jobs is 1
4. In case of qsub --venode=4, the number of jobs is 1

The VE resources rounded up to a multiple of the default number of incorporated VE nodes are allocated per job. As a result, if the total number of VEs cannot be divided by the default number of incorporated VE nodes as shown in the above **2.** and **4.**, the VE resources on the execution host are left.

### 12.3. Limit of the range of the total number of VEs that can be entered

This function is available only for the environment whose execution host is SX-Aurora TSUBASA system.

The user can specify the range of the total number of VE nodes (upper and lower limit values) that can be entered by specifying the total number of VE nodes (with qsub --venode) per queue. If the total number of VEs specified with qsub is out of this range, entering a request results in an error.
This limitation is only effective for the requests entered with the total number of VE nodes (with qsub --venode) specified. By default, it does not limit requests submitted with the number of VE nodes per logical host (qsub --venum-lhost). To limit them together, see **Extended submit limit for the number of VE nodes**.

This limitation is ignored for the requests entered with the limit value for the number of VEs and the number of jobs per logical host specified.

Specify the range of the total number of VE nodes by using the qmgr subcommands below. This setting requires administrator rights.

```
set execution_queue submit_venode_range = (<min>,<max>) <queue-name>
set interactive_queue submit_venode_range = (<min>,<max>) <queue-name>
```

For <min>, specify an integer of 0 or larger and <max> or smaller. If you specify the same value for <min> and <max>, you can only specify that value for qsub --venode. If you specify 0 for <min> and <max>, you can only enter requests that do not use VE nodes.
<min> is set to 0 and <max> is set to UNLIMITED immediately after a queue is created.

You can check the specified range limiting value as follows, by using the qstat -Qf command.

```
$ qstat -Qf bq
Execution Queue:bq@bsv1
    Run State    = Active
    Submit State = Enable
    ...
Defined VE Number     = 1
Submit VE Node Range (min,max)  = 0,UNLIMITED
Total Request         = 0
Arriving Request      = 0
    ...
```

### Extended submit limit for the number of VE nodes

To apply the queue VE total number range limit to the value obtained by multiplying the number of VE nodes per logical host (qsub --venum-lhost) by the number of jobs to be run (qsub -b), configure the following.

Configure the nqsd.conf file (/etc/opt/nec/nqsv/nqsd.conf) in the following format. It is then enabled by restarting the batch server or by re-loading the configuration file by running the "load nqsd_conf" subcommand in qmgr.

| queues_for_extended_venode_range [*queue* ...] |
|---|

Multiple queues can be specified separated by spaces. Enable this function for the queue specified in *queue*. If *queue* is not specified, this function is enabled for all queues present in BSV.
If the queues_for_extended_venode_range line is missing, there is no limit for requests submitted by specifying the number of VE nodes for each logical host (qsub --venum-lhost).

### 12.4. HCA Assignment

This function is available only for the environment whose execution host is SX-Aurora TSUBASA system.
When execution host has VE and HCA, NQSV can assign appropriate HCA's port to a job.
For details of HCA assignment, please refer to [JobManipulator].

### 12.5. HCA failure check

BSV can remove the JSV from operation when detect the failure of HCA if VE and HCA are installed on the execution host. The timing of detecting the failure is PRE-RUNNING and POST-RUNNING state of every request.

This function judges the system status healthy when all HCA on the execution host are active. If even one HCA is inactive, the system is treated as unhealthy. This function uses ibstat command which is installed on the execution host to detect the HCA failure. Therefore this function not available on the host that ibstat command is not installed.

It is able to configure the action for every JSV when HCA failure detected.

| | |
|---|---|
| off | Do nothing when HCA failure detected. |
| unbind | Unbind the JSV from queue when HCA failure detected. If there is running jobs on the execution host, which execution continue. |
| down | Stop JSV when HCA failure detected. If there is running jobs on the execution |

| | host, it becomes stall status. |
|---|---|

To set the action of JSV when HCA failure detected, use following sub-command of qmgr.

```
set job_server hca_failure_check = {off | down | unbind} job_server_id = jsvid
```

It configuration can be displayed by qstat -Sf command.

```
$ qstat -Sf
Job Server Name: JobServer0001
    ...
    HCA Failure Check = UNBIND
    ...
```

### 12.6. Concurrent GPU Number Limit

It is possible to set concurrent GPU number limit for jobs per queue regarding requests submitted to batch queue and interactive queue.
Batch requests submitted will be rejected by the batch queue if resource consumption specified to the batch request exceeds resource limit set to the batch queue.

In order to set GPU number limit to queues, the sub-command of qmgr is to be used as follows.

```
set execution_queue per_job gpu_number_limit = limit  queue
set interactive_queue per_job gpu_number_limit = limit  queue
```

The operator privilege or higher is necessary to set the limit.
Value between 0 and 256 or UNLIMITED can be set to GPU number limit.

In addition, in order to set resource default of concurrent GPUs for queues, the following sub-command of qmgr is to be used.

```
set execution_queue standard per_job gpu_number_limit = limit  queue
set interactive_queue standard per_job gpu_number_limit = limit  queue
```

Resource default of concurrent GPUs is 0.
Note that it is different from other resource default, UNLIMITED.

Resource limits and default set for queue can be checked with qstat(1) -Q -f.

### 12.7. Automatic switching of VE NUMA mode

This function is available only when execution hosts are SX-Aurora TSUBASA.

In case of execution jobs with VE, the performance of the application may be improved if the partitioning mode for the VE is changed. This mode can be changed to the normal mode or the partitioning mode by the command of VEOS. Please refer "SX-Aurora TSUBASA VEOS NUMA Mode Guide for Partitioning Mode" for detail of the partitioning mode / NUMA mode.

To use this function, you need to set a shell script (/opt/nec/nqsv/sbin/venuma_chg.sh) to switch partitioning modes on the execution host.
This script is named /opt/nec/nqsv/sbin/venuma_chg.sh.sample when installing NQSV. Change this to /opt/nec/nqsv/sbin/venuma_chg.sh.
In the script, there is a variable called DEFAULT (ON or OFF) that sets the default value for the partitioning mode of the production environment. Set DEFAULT ON or OFF according to your operating environment. The default value is OFF.

In case of NQSV, it is to specify ON or OFF of VE NUMA mode by qsub --venuma at submitting requests. NQSV will automatically switch VE NUMA to the partitioning mode by the command of VEOS if necessary. Changing the partitioning mode for the assigned VE to the jobs is switched appropriate. Jobs execute in the default setting if VE NUMA mode isn't specified at submitting requests.

Use the --venuma option of qsub(1) to submit a request.

```
qsub --venuma={on|off}
```

This setting can be confirmed in "VE NUMA Mode" of the command qstat -f. The command qalter cannot change this setting after submitting requests. Interactive requests are not have this setting.

If failure of the changing the mode or not completion of the changing the mode in 90 seconds, the request fails to execution and becomes "QUEUED" state. If you want to change the wait time to switch the partitioning mode except 90 seconds, the environment variable NQSV_VENUMA_TIMEOUT is set at starting the job server.

**[Remarks]**

- It takes time to switch the partitioning mode. You may measure the time it takes to execute in your actual environment, and you should set that time as the elapse margin time of the queue to be submit requests by smgr(1M). Please refer NQSV User's Guide [JobManipulator].

- NQSV changes the partitioning mode only at starting the request. Setting of the partitioning mode may be different for the default value after terminating the request.
- This function and Suspending VE jobs to run urgent requests cannot be used at the same time. For detail on urgent requests, please refer NQSV User's Guide [JobManipulator].

## 12.8. Configuration for Multi-instance GPU (MIG)

### 12.8.1. About Multi-instance GPU (MIG)

Multi-instance GPU (hereinafter referred to as MIG) is a feature on some NVIDIA GPUs (A30, A100, H100, etc.) that allows physical resource partitioning within the GPU.
When using MIG, qsub option "--gpunum-lhost" and various GPU-related functions cannot be used because access to GPU resources is different from conventional GPUs. Therefore, qsub option "--mig" is used to allocate resources.

### 12.8.2. How to use MIG

For example, in the case of NVIDIA's H100, up to 8 GPUs can be installed on one host. It is then possible to enable or disable the MIG function for each GPU.

This function is automatically enabled only when all of the following conditions are met.

- A MIG-compatible GPU card is installed on the execution host.
- The nvidia-smi command is installed.
- GPU instance (GI) of MIG is already configured.

You can check whether the MIG functionality is enabled for each execution host by looking for "Multi Instances GPU" in the output of the qstat -Ef command (execution host information).

[Notes]
If MIG is enabled, the custom resources are created by appending "M" to the begining of the GI profile names of the execution hosts. Do not modify these custom resources because NQSV system uses them.

# 13. Hook Script Function

The hook script function executes a script defined by an administrator (called a hook script) on a batch server host when a request transits to a certain state. A hook script can be defined in any state to which a request may transit.

## 13.1. Save a hook script

To locate a hook script, create a directory with a queue name in the following directory of a batch server host in advance. Then locate the script with a name indicating the request state in the created directory.

```
/opt/nec/nqsv/sbin/hook_prog
```

The following table shows the hook script naming rule.

| Request state in which a hook script is started | Hook script name |
|---|---|
| QUEUED | request_queued.sh |
| WAITING | request_waiting.sh |
| HELD | request_held.sh |
| HOLDING | request_holding.sh |
| SUSPENDING | request_suspending.sh |
| SUSPENDED | request_suspended.sh |
| ARRIVING | request_arriving.sh |
| TRANSITING | request_transiting.sh |
| STAGING | request_staging.sh |
| PRE-RUNNING | request_prerunning.sh |
| RUNNING | request_running.sh |
| POST-RUNNING | request_postrunning.sh |
| EXITING | request_exiting.sh |
| MIGRATING | request_migrating.sh |
| MOVED | request_moved.sh |
| EXITED | request_exited.sh |
| RESUMING | request_resuming.sh |

The following shows a path name example of a hook script to be executed when a request submitted in the batch1 queue transits to the RUNNING state.

```
/opt/nec/nqsv/sbin/hook_prog/batch1/request_running.sh
```

### 13.2. Enabling and disabling a hook script

A hook script is executed only when the hook script function is enabled for a queue (batch queue, interactive queue, routing queue). This script is not executed only by saving it in the above mentioned directory.
Use a subcommand of qmgr(1M) to enable the hook script function. The operator privilege is required to configure this setting.

| Queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | set execution_queue hook_function |
| Interactive queue | set interactive_queue hook_function |
| Routing queue | set routing_queue hook_function |

The following shows a setting example to enable the hook script function for batch queue batch1.

```
$ qmgr -Po
Mgr: set execution_queue hook_function on batch1
```

To check whether the hook script function is enabled, execute the qstat(1) command with -Q -f specified. Then the following is displayed.

```
$ qstat -Qf
Execution Queue: batch1@bsv1
    Run State    = Inactive
      :
    IntelMPI Process Manager = hydra
    Hook function = ON
      :
```

### 13.3. Executing hook script

When the hook script function is enabled for a queue and a hook script is located in a predetermined directory, a batch server executes the hook script according to the state transition of a request submitted in the queue.
Unlike the user EXIT function, the hook script function does not wait for completion of a hook script while the request state is transitioning. In addition, there is no difference in the request state transition behavior according to the script execution result (exit status).

When a hook script is executed, the following environment variables are set.

| Environment variable | Explanation | Value |
|---|---|---|
| HOOK_REASON | Cause of the request state | RUN |

| | | |
|---|---|---|
| | transition | DELETE<br>PRERUN_FAIL etc |
| HOOK_RSTPREV | Previous request state | QUEUED<br>PRERUNNING<br>RUNNING etc. |
| HOOK_RID | Request ID | For a normal request:<br><sequence-number>.<host-name><br>For a parametric request:<br><sequence-number>[].<host-name> |
| HOOK_USER | Request owner name | User name. |
| HOOK_GROUP | Name of the group to which the request owner belongs | Group name. |
| HOOK_HOME | Home directory of the request owner | Absolute path name of the home directory |
| HOOK_QUEUE | Name of the queue to which a request was submitted | Queue name. |
| HOOK_NJOBS | Number of jobs in the request | Number of jobs in the request. |
| Other settings conform to the environment variables set to the request. | | |

## 14. User Pre-Post Script Function

The User Pre-Post script function executes a script specified when submitting a request (called a UserPP script) before job execution (PRE-RUNNING) or after job execution (POST-RUNNING).

A request owner can create and specify a UserPP script. For example, using this function, you can check the health of a node before starting a job execution and also clear temporary files in an execution host after the job execution. A UserPP script is not subject to resource limitation and accounting.

A timeout occurrence can be monitored to prevent a UserPP script from being stalled and resources from being occupied due to execution of an invalid script. To monitor a timeout occurrence, set a timeout time of a UserPP script to a queue. If the timeout time set to the queue has elapsed after the UserPP script execution started, the script execution stops.

### 14.1. Setting a timeout time of a UserPP script

By setting a timeout time of a UserPP script, a KILL signal is sent to the UserPP script if an execution error such as a stall occurs and execution of the UserPP script is forcibly ended.

Use a subcommand of qmgr(1M) to set a timeout time. The following shows an example to set a timeout time of a UserPP script to 900 seconds for batch queue exec1. Execute qmgr(1M) with the operator privilege.

```
$ qmgr -Po
Mgr: set execution_queue userpp_timeout = 900 exec1
Set UserPP Timeout: exec1
```

The qmgr(1M) subcommands that are used to configure each setting and their default value is as follows.

| Value | qmgr(1M) sub-command | Default |
|---|---|---|
| Timeout time of a UserPP script (in seconds) | set execution_queue userpp_timeout<br>set interactive_queue userpp_timeout | 300 |

If 0 (zero) is specified, timeout monitoring will not be performed.

# 15. Provisioning environment in conjunction with OpenStack

This function is NOT available for the environment whose execution host is SX-Aurora TSUBASA system.

NQSV can dynamically configure a job execution environment in an execution host in conjunction with OpenStack. To carry out a job, this function can dynamically switch an environment (OS version, installed libraries, ISV, etc.) to the environment necessary for the job in the relevant host.

Provisioning by using a virtual machine (VM) and provisioning by using a bare metal server are supported as a provisioning environment in conjunction with OpenStack. The function is also provided to define and manage an OS image and resource information of this provisioning environment as a template.

## 15.1. Configuring a provisioning environment by using a virtual machine

NQSV can dynamically configure a job execution environment in an execution host in conjunction with the OpenStack virtual machine instance creation function.

This section describes the conjunction between NQSV and the OpenStack virtual machine instance creation function using OpenStack of the following version as an example.

- OpenStack Liberty (October 2015), Community edition

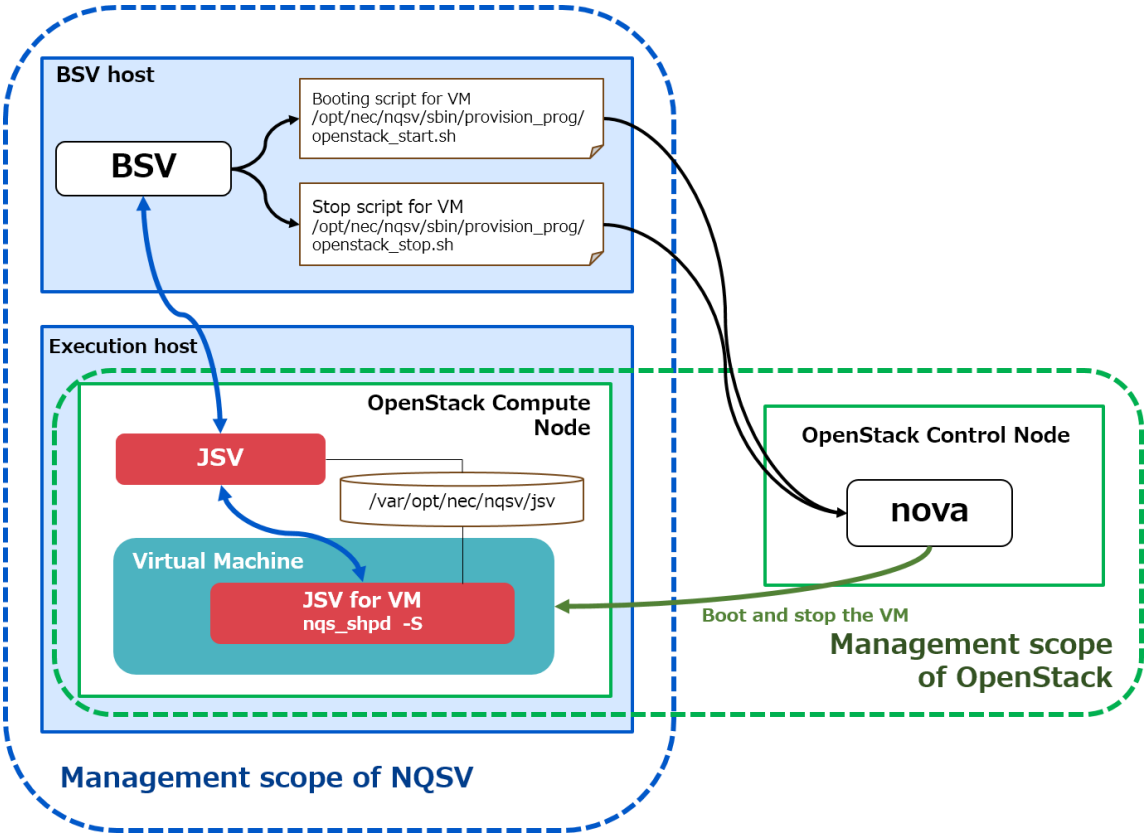


Figure 15-1 : Conceptual diagram of NQSV and OpenStack (Virtual machine)

In the above example, an NQSV execution host is used as a Compute node of OpenStack. When a job starts, NQSV executes a virtual machine startup script from a BSV to instruct a Control node of OpenStack to start a virtual machine. The virtual machine is started by the nova command and the job is executed in the started virtual machine with a JSV in the virtual machine and a JSV on the execution host working together. When the job ends, the BSV executes a virtual machine stop script to instruct the Control node to stop the virtual machine.

### 15.1.1. OpenStack environment setting

To make OpenStack work together with NQSV, configure an OpenStack environment as follows in advance.

(1) Using the nova command from a BSV host

Configure an environment so that the root user can use the nova commands for a Control node of OpenStack.

(2) Setting an execution host as a Compute node of OpenStack

Set an NQSV execution host as a Compute node of OpenStack.

(3) Setting up a virtual machine boot image

Set up a virtual machine boot image to satisfy the following conditions:

- A JSV package is installed in a virtual machine boot image. The boot image is set up to start a job server automatically by an init script in the **nqs_shpd -S** format when an OS starts. <u>It is not necessary to specify other nqs_shpd options (such as -h and -n).</u>
- A virtual machine OS and an execution host OS can communicate with each other.
- A virtual machine OS and an execution host OS share a JSVDB (/var/opt/nec/nqsv/jsv).
- Virtual machines started on execution hosts can communicate with each other by using a host name.
- An account of a job submit user can be used on a virtual machine OS.

### 15.1.2. Configuring a job server on an execution host

A job server (JSV) that has started on a virtual machine will connect to a JSV on an execution host. The JSV on the virtual machine uses a host name acquired by the JSV on the execution host as its own name at connection.

But, when being different in the host name to which I can refer on the actual host and on the virtual machine, please designate the host name seen from the virtual machine side specifically in -H option at the time of a JSV start on the execution host.
JSV start option on the execution host:

```
nqs_shpd -H <vm_host> -h <bsv_host> -n <jsvno>
```

This enables to connect the JSV on the virtual machine to the JSV on the execution host by using the host name <vm_host> that the virtual machine can reference.

### 15.1.3. Virtual machine startup script

A virtual machine is started by a shell script created by a system administrator. Locate a shell script in the following path.

Startup script : **/opt/nec/nqsv/sbin/provision_prog/openstack_start.sh**

Create a startup script to execute the following processes according to the NQSV and OpenStack execution environments.

(1) Processes to be executed in a virtual machine startup script

The following three processes must be executed in a virtual machine startup script.

1) **Starting a virtual machine**

Process to execute the nova commands according to the environment variables specified in the script to start a virtual machine.

2) **Waiting for all virtual machines to start**

Process to wait until all virtual machines are fully started.

3) **Creating a virtual machine information file (VMIP)**

Process to describe the following items by delimiting with a space in the file specified by the environment variable NQS_OPENSTACK_IPINFOPATH. Items of one job must be described in one line. (This is called a VMIP file.)

- Started job numbers
- IP addresses assigned to virtual machines
- IDs of started virtual machines

[sample form]

```
0 192.168.1.101 8a850692-0362-425d-b9af-f1cb20a450c0
1 192.168.1.102 9d52de61-a439-46cc-89ee-dd4a66eaafb6
```

When the above three processes are complete, the script terminates with **exit 0**.

If starting a virtual machine failed, for example, due to a failure of the nova command during the processes 1) to 3) above, the relevant job number is described in the file specified by the environment variable NQS_OPENSTACK_FAILINFOPATH. A job number of one job is described in one line. (This is called a VMFAIL file.) Then the script terminates with **an exit code other than exit 0**.

(2) Environment variables to execute a virtual machine startup script

Specify the following environment variables to execute a virtual machine startup script.

| Environment variable | Value |
|---|---|
| NQS_OPENSTACK_TEMPLATE_NAME | Template name |
| NQS_OPENSTACK_IMAGE_NAME | OS image name |
| NQS_OPENSTACK_CPU | Number of CPUs |
| NQS_OPENSTACK_GPU | Number of GPUs |
| NQS_OPENSTACK_MEMORY | Memory size<br>(Example of "Size" and "Unit": 100 MB) |
| NQS_OPENSTACK_FLAVOR | Flavor name |
| NQS_OPENSTACK_CUSTOM | Custom definition |
| NQS_OPENSTACK_QUE | Name of the queue to which a request was submitted |
| NQS_OPENSTACK_RID | Request ID<br>Format: <seqno>.<mid><br>For <mid>, specify a number. |
| NQS_OPENSTACK_IPINFOPATH | Path in which to save a VMIP file |
| NQS_OPENSTACK_FAILINFOPATH | Path in which to save a VMFAIL file |
| NQS_OPENSTACK_PROCTYPE | "EXECUTION" (For forward processing)<br>"ROLLBACK" (For rollback) |
| NQS_OPENSTACK_HOSTS | Host name and job number of a virtual machine to be started<br>Format:<br><jobno>:<hostname>[ <jobno>:<hostname> ...]<br>* Use a space as a delimiter |

(3) Rollback processing

If executing a virtual machine startup script failed (that is, the script ended with an exit code other than exit 0), the script will be executed again to perform a rollback processing. The environment variable NQS_OPENSTACK_PROCTYPE is used to determine whether to execute a forward processing or rollback processing.

In a rollback processing, it is required to stop virtual machines with IDs described in the VMIP file and wait for all of them to stop completely. NQSV executes the subsequent processes after

the rollback processing assuming that all the virtual machines have stopped. For the host that cannot be stopped in the rollback processing, the relevant job number must be described in a VMFAIL file.

(4) Timeout

Timeout monitoring is performed while a virtual machine startup script is executed. The value of the estimated startup time (Boot Timeout) set in a template is used as a timeout time.

In a forward processing of a virtual machine startup script, if the script does not terminate even if the time set to Boot Timeout has passed, it is assumed that executing the startup script failed. Then the virtual machine startup script is interrupted (by sending a KILL signal) and a rollback processing is performed.

In a rollback processing, timeout monitoring is also performed. If the processing does not terminate even if the time set to Boot Timeout has passed, executing the virtual machine startup script (rollback processing) is interrupted. However, if a rollback processing is interrupted due to a timeout, a virtual machine may remain on an execution host without stopping.

(5) Notes on creating a virtual machine startup script

When creating a virtual machine startup script to start multiple virtual machines, select either of the following two operation types to be performed at detection of startup failure. The subsequent scheduling operation differs depending on which operation type is selected. Since there are advantages and disadvantages to both operation types, select either according to your operations.

1) **Waiting until the startup processes of all virtual machines are complete and detecting all the virtual machines that could not start.**

   This operation can detect all the virtual machines that could not start among the virtual machines to start. The advantage of this operation is that a schedule can be created to avoid the relevant host and assign a request to another host in the subsequent scheduling operation.

   On the other hand, since it is required to wait until all target virtual machines are fully started, it takes time to assign the request again after the detection of the first startup failure.

2) **Interrupting a process assuming that all startup processes failed if any virtual machine startup failure is detected.**

   The advantage of this operation is that an action such as reassignment of a request can be performed immediately because a virtual machine startup failure can be detected as soon as its occurrence.

On the other hand, if another virtual machine cannot start after the detection of the first startup failure, that startup failure cannot be detected. Therefore, when the request is assigned to the relevant host next time, a startup failure may occur again.

### 15.1.4. Virtual machine stop script

A virtual machine is stopped by a shell script created by a system administrator. Locate a shell script in the following path.

Stop Script： **/opt/nec/nqsv/sbin/provision_prog/openstack_stop.sh**

Create a stop script to execute the following processes according to the NQSV and OpenStack execution environments.

(1) Processes to be executed in a virtual machine stop script

The following two processes must be executed in a virtual machine stop script.

1) **Stopping a virtual machine**

Process to execute the nova commands according to the environment variables specified in the script to stop a virtual machine.

2) **Waiting for all virtual machines to stop**

Process to wait until all virtual machines are fully stopped (or until it is found that virtual machines cannot be stopped).

* Even if a stop script fails, a rollback processing is not performed. Therefore, be sure to confirm that virtual machines have stopped completely. NQSV executes the subsequent processes assuming that all the virtual machines have completely stopped when the execution of the virtual machine stop script is complete.

When the above two processes are complete, the script terminates with **exit 0**.

If stopping a virtual machine failed due to an error occurrence while waiting for the virtual machines to stop, the relevant job number is described in the VMFAIL file specified by the environment variable NQS_OPENSTACK_FAILINFOPATH. A job number of one job is described in one line. Then the script terminates with **an exit code other than exit 0**.

(2) Environment variables to execute a virtual machine stop script

Specify the following environment variables to execute a virtual machine stop script.

| Environment variable | Value |
|---|---|
| NQS_OPENSTACK_TEMPLATE_NAME | Template name |
| NQS_OPENSTACK_IMAGE_NAME | OS image name |
| NQS_OPENSTACK_CPU | Number of CPUs |
| NQS_OPENSTACK_GPU | Number of GPUs |

| | |
|---|---|
| NQS_OPENSTACK_MEMORY | Memory size<br>(Example of "Size" and "Unit": 100 MB) |
| NQS_OPENSTACK_FLAVOR | Flavor name |
| NQS_OPENSTACK_CUSTOM | Custom definition |
| NQS_OPENSTACK_QUE | Name of the queue to which a request was submitted |
| NQS_OPENSTACK_RID | Request ID<br>Format: <seqno>.<mid><br>For <mid>, specify a number. |
| NQS_OPENSTACK_IPINFOPATH | Path in which to save a VMIP file |
| NQS_OPENSTACK_FAILINFOPATH | Path in which to save a VMFAIL file |
| NQS_OPENSTACK_IPADDRS | List of job numbers and IP addresses assigned to virtual machines. The format is as follows:<br>Format:<br><jobno>:<ip addr> [<jobno>:<ip addr> ]<br>* Use a space as a delimiter<br>The format of <ip addr> is XXX.YYY.ZZZ.NNN.<br>(Example: 172.16.1.1) |
| NQS_OPENSTACK_PROCTYPE | "EXECUTION" |
| NQS_OPENSTACK_HOSTS | Host name and job number of a virtual machine to be stopped<br>Format:<br><jobno>:<hostname>[ <jobno>:<hostname> ...]<br>* Use a space as a delimiter |

(3) Timeout

Timeout monitoring is performed while a virtual machine stop script is executed. The value of the estimated stop time (Stop Timeout) set in a template is used as a timeout time.

If the virtual machine stop script does not terminate even if the time set to Stop Timeout has passed, it is assumed that executing the script failed. Then the virtual machine stop script is interrupted (by sending a KILL signal).

If the virtual machine stop script is interrupted due to a timeout, a virtual machine may remain without stopping.

### 15.1.5. Sample virtual machine startup and stop scripts

The sample virtual machine startup and stop scripts are installed in a BSV host. By referring these sample scripts, create a virtual machine startup and stop scripts including the above described processes and locate them in an appropriate path.

[Notes]
These sample scripts offer an implementation image of the minimum function. They do not guarantee operations on all OpenStack environments. When configuring an environment, implement appropriate processes according to your actual environment.

The summary of each script is described below.

(1) Sample virtual machine startup script

Installation path: /opt/nec/nqsv/sbin/provision_prog/openstack_start.sh.sample

Process overview:

When EXECUTION is set to the environment variable NQS_OPENSTACK_PROCTYPE:

- The nova boot command starts a virtual machine on the host specified for the environment variable NQS_OPENSTACK_HOSTS. A free floating-ip acquired by nova floating-ip-list in advance is assigned to the started virtual machine. The --availability-zone option of the nova boot command expressly specifies the host on which to start a virtual machine.
- The nova show command periodically monitors the instance ID acquired from the startup message output when executing nova boot and waits until the status will not be BUILD.

When ROLLBACK is set to the environment variable NQS_OPENSTACK_PROCTYPE:

- The nova delete command stops a virtual machine with the instance ID acquired from the VMIP file specified for the environment variable NQS_OPENSTACK_IPINFOPATH.
- The process waits until the nova show command will not be able to reference information of the relevant instance ID.

(2) Sample virtual machine stop script

Installation path: /opt/nec/nqsv/sbin/provision_prog/openstack_stop.sh.sample

Process overview:

- The nova list command acquires an instance ID according to the IP address acquired from the file specified for the environment variable NQS_OPENSTACK_IPADDRS. The nova delete command stops a virtual machine with the acquired instance ID.
- The process waits until the nova show command will not be able to reference information of the relevant instance.

### 15.1.6. Incorporating a virtual machine to NQSV

A virtual machine can be incorporated in a job operation by binding a job server of an execution host, on which to start a virtual machine that is set as an OpenStack Compute node to a queue. By unbinding the relevant job server from the queue, the virtual machine can be removed from a job operation.

> [Notes]
> When binding an execution host on which to start a virtual machine to a queue, it is not possible to coexist that execution host with a bare metal server and usual execution host in the queue.
> The queue to which to bind an execution host on which to start a virtual machine must be dedicated to a virtual machine.

## 15.2. Configuring a provisioning environment by using a bare metal server

NQSV can dynamically configure a job execution environment using a bare metal server as an execution host in conjunction with the OpenStack bare metal provisioning function.

This section describes the conjunction between NQSV and the OpenStack bare metal provisioning function using OpenStack of the following version as an example.

- Red Hat OpenStack Platform (RHOSP) 8

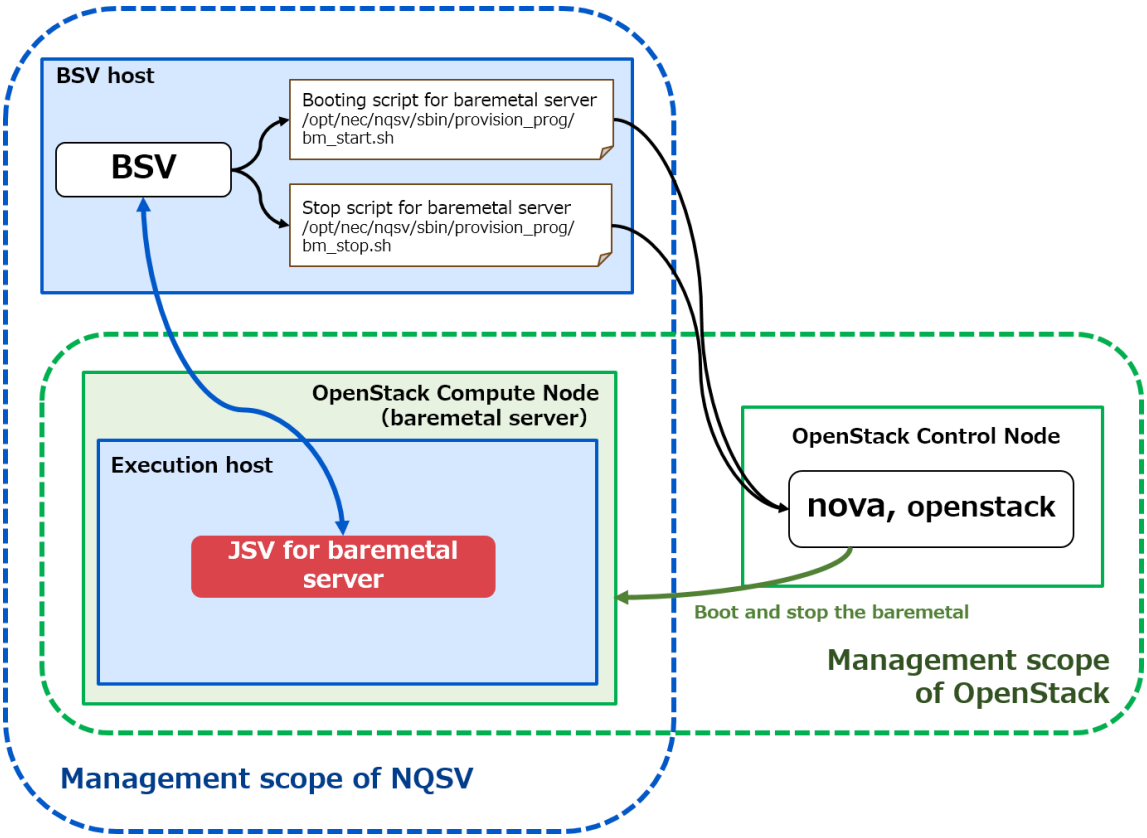



Figure 15-2 : Conceptual diagram of NQSV and OpenStack (Baremetal)

An OpenStack Compute node (bare metal server) is registered as an NQSV execution host. When a job starts, NQSV executes a bare metal server startup script from a BSV to instruct a Control node of OpenStack to start a bare metal server. The bare metal server is started by the nova command and the job is executed by using the started bare metal server as an NQSV execution host. When a job ends, the BSV executes a bare metal server stop script to instruct the Control node to stop the bare metal server. OpenStack stops the bare metal server.

### 15.2.1. OpenStack environment setting

To make OpenStack work together with NQSV, configure an OpenStack environment as follows in advance.

(1) Using the nova command from a BSV host

Configure an environment so that the root user can use the nova and OpenStack commands for a Control node of OpenStack.

(2) Setting up a bare metal server boot image

Set up a bare metal server boot image to satisfy the following conditions:

- A JSV package is installed in a bare metal server boot image. The boot image is set up to start a job server automatically by an init script in the **nqs_shpd -B <BSV_host>** format when an OS starts. It is not necessary to specify other nqs_shpd options (such as -h and -n).
- Bare metal servers can communicate with each other by using a host name.
- Bare metal servers can communicate with the BSV host by using a host name.
- An account of a job submit user can be used on a bare metal server OS.

### 15.2.2. Bare metal server startup script

A bare metal server is started by a shell script created by a system administrator. Locate a shell script in the following path.

Startup script : **/opt/nec/nqsv/sbin/provision_prog/bm_start.sh**

Create a startup script to execute the following processes according to the NQSV and OpenStack execution environments.

(1) Processes to be executed in a bare metal server startup script

The following two processes must be executed in a bare metal server startup script.

1) **Starting a bare metal server**

Process to execute the nova commands according to the environment variables specified in the script to start a bare metal server.

2) **Waiting for bare metal servers to start**

Process to wait until bare metal servers are fully started.

When the above two processes are complete, the script terminates with **exit 0**.

If starting a bare metal server failed, for example, due to a failure of the nova command during the processes **1)** to **2)** above, the script terminates with an exit code other than **exit 0**.

(2) Environment variables to execute a bare metal server startup script

Specify the following environment variables to execute a bare metal server startup script.

| Environment variable | Value |
|---|---|
| NQS_BM_TEMPLATE_NAME | Template name |
| NQS_BM_IMAGE_NAME | OS image name |
| NQS_BM_CPU | Number of CPUs |
| NQS_BM_GPU | Number of GPUs |
| NQS_BM_MEMORY | Memory size<br>(Example of "Size" and "Unit": 100 MB) |
| NQS_BM_FLAVOR | Flavor name |
| NQS_BM_CUSTOM | Custom definition |
| NQS_BM_PROCTYPE | "EXECUTION" (For forward processing)<br>"ROLLBACK" (For rollback) |
| NQS_ BM _HOSTS | Host name a bare metal server to be started<br>Format:<br><hostname>[ <hostname> ...]<br>* Use a space as a delimiter |

(3) Rollback processing

If executing a bare metal server startup script failed (that is, the script ended with an exit code other than "exit 0"), the script will be executed again to perform a rollback processing. The environment variable NQS_OPENSTACK_PROCTYPE is used to determine whether to execute a forward processing or rollback processing.

In a rollback processing, it is required to stop the host specified for the environment variable NQS_BM_HOSTS and wait for all the bare metal servers to stop completely. NQSV executes the subsequent processes after the rollback processing assuming that all the bare metal servers have stopped.

(4) Timeout

Timeout monitoring is performed while a bare metal server startup script is executed. The value of the estimated startup time (Boot Timeout) set in a template is used as a timeout time.

In a forward processing of a bare metal server startup script, if the script does not terminate even if the time set to Boot Timeout has passed, it is assumed that executing the startup script failed. Then the bare metal server startup script is interrupted (by sending a KILL signal) and a rollback processing is performed.

In a rollback processing, timeout monitoring is also performed. If the processing does not terminate even if the time set to Boot Timeout has passed, executing the bare metal server startup script is interrupted. However, if a rollback processing is interrupted due to a timeout, a bare metal server may remain without stopping.

### 15.2.3. Bare metal server stop script

A bare metal server is stopped by a shell script created by a system administrator. Locate a shell script in the following path.

Stop Script : **/opt/nec/nqsv/sbin/provision_prog/bm_stop.sh for stops**

Create a stop script to execute the following processes according to the NQSV and OpenStack execution environments.

(1) Processes to be executed in a bare metal server stop script

The following two processes must be executed in a bare metal server stop script.

1) **Stopping a bare metal server**

Process to execute the nova commands according to the environment variables specified in the script to stop a bare metal server.

2) **Waiting for bare metal servers to stop**

Process to wait until all bare metal servers are fully stopped (or until it is found that bare metal servers cannot be stopped).

***** Even if a stop script fails, a rollback processing is not performed. Therefore, be sure to confirm that bare metal servers have stopped completely. NQSV executes the subsequent processes assuming that all the bare metal servers have completely stopped when the execution of the bare metal server stop script is complete.

When the above two processes are complete, the script terminates with **exit 0**.
If any bare metal server cannot be stopped because of failure occurrence while waiting for bare metal servers to stop, the script terminates with an **exit code other than exit 0**.

(2) Environment variables to execute a bare metal server stop script

Specify the following environment variables to execute a bare metal server stop script.

| Environment variable | Value |
|---|---|
| NQS_BM_TEMPLATE_NAME | Template name |
| NQS_BM_IMAGE_NAME | OS image name |
| NQS_BM_CPU | Number of CPUs |
| NQS_BM_GPU | Number of GPUs |
| NQS_BM_MEMORY | Memory size<br>(Example of "Size" and "Unit": 100 MB) |
| NQS_BM_FLAVOR | Flavor name |
| NQS_BM_CUSTOM | Custom definition |
| NQS_BM_PROCTYPE | "EXECUTION" |
| NQS_ BM _HOSTS | Host name a bare metal server to be stopped<br>Format: |

| | |
|---|---|
| | <hostname>[ <hostname> ...]<br>* Use a space as a delimiter |

(3) Timeout

Timeout monitoring is performed while a bare metal server stop script is executed. The value of the estimated stop time (Stop Timeout) set in a template is used as a timeout time.
If the bare metal server stop script does not terminate even if the time set to Stop Timeout has passed, it is assumed that executing the script failed. Then the bare metal server stop script is interrupted (by sending a KILL signal).
If the bare metal server stop script is interrupted due to a timeout, a bare metal server may remain without stopping.

### 15.2.4. Sample bare metal server startup and stop scripts

The sample bare metal server startup and stop scripts are installed in a BSV host. By referring these sample scripts, create a bare metal server startup and stop scripts including the above described processes and locate them in an appropriate path.

> [Notes]
> These sample scripts offer an implementation image of the minimum function. They do not guarantee operations on all OpenStack environments. When configuring an environment, implement appropriate processes according to your actual environment.

The summary of each script is described below.

(1) Sample bare metal server startup script

Installation path: /opt/nec/nqsv/sbin/provision_prog/bm_start.sh.sample

Process overview:

When EXECUTION is set to the environment variable NQS_BM_PROCTYPE:

- The nova boot command starts a bare metal server specified for the environment variable NQS_BM_HOSTS.
- The nova show command periodically monitors the instance ID acquired from the startup message output when executing nova boot and waits until the status will not be BUILD.

When ROLLBACK is set to the environment variable NQS_BM_PROCTYPE:

- The nova list command acquires an instance ID according to the host name specified for the environment variable NQS_BM_HOSTS. The nova delete command stops a bare metal server with the acquired instance ID.

- The process waits until the nova show command will not be able to reference information of the relevant instance.

(2) Sample bare metal server stop script

Installation path： /opt/nec/nqsv/sbin/provision_prog/bm_stop.sh.sample

Process overview:

- The nova list command acquires an instance ID according to the host name specified for the environment variable NQS_BM_HOSTS. The nova delete command stops a bare metal server with the acquired instance ID.
- The process waits until the nova show command will not be able to reference information of the relevant instance.

### 15.2.5. Incorporating a bare metal server to NQSV

To use a bare metal server environment, it is necessary to register a bare metal server as an execution host to NQSV in advance.

The registered bare metal server can be incorporated in a job operation by binding it to a queue in the same way as incorporating a usual execution host.

(1) Registering a bare metal server

Use the attach baremetal_host subcommand of qmgr(1M) to register a bare metal server. When registering a bare metal server, the number of CPUs, memory size, and number of GPUs must be registered in addition to the host name and job server number.

Specification of the attach baremetal_host command (The administrator privilege is required to use this command.)

```
attach baremetal_host host = <host_name> job_server_id=<jsv_id> cpu = <cpunum>
memory=<memsz> gpu=<gpunum>
```

As with the attach execution_host subcommand, it is possible to register multiple bare metal servers at once. To do so, create a list file of their host names, job server numbers, number of CPUs, memory sizes, and number of GPUs in advance and specify the created file for the subcommand.

In a list file, one line must include information of one bare metal server, such as a host name, job server number, number of CPUs, memory size, and number of GPUs that are separated by a space. Describe lines as many as the number of bare metal servers.

```
Host1 0 4 10GB 0
Host2 1 4 20GB 1
```

Use the following subcommand to register bare metal servers at once by specifying the above list file.

```
attach baremetal_host file=<file_path>
```

> [Notes]
> Before starting the operation, start the JSV manually on a bare metal host and set it to LINKUP state to register the license information to be used by the bare metal host to the BSV.
> When starting the JSV, specify the JSVID registered by using the attach baremetal_host subcommand of qmgr.

(2) Removing a bare metal server

Use the detach baremetal_host subcommand of qmgr(1M) to delete a registered bare metal server. A bare metal server can be removed when no job uses the job server of the target bare metal server and the state of the job server is LINKDOWN.

Specification of the detach baremetal_host command (The administrator privilege is required to use this command.)

```
detach baremetal_host host = <host_name>
detach baremetal_host host = (<host_name>, <host_name>, ...)
detach baremetal_host job_server_id = <jsv_id>
detach baremetal_host job_server_id = (<jsv_id1>,<jsv_id2>, ...)
detach baremetal_host job_server_id = < jsv_id1>-< jsv_id2>
detach baremetal_host all
```

(3) Referencing the registered bare metal server information

Use qstat -E or qstat -Et to reference the information of the registered bare metal server in the same way as referencing a usual execution host. [B] at the beginning of an ExecutionHost item indicates that the line shows bare metal server information.

[Indication example]

```
$ qstat -Et
ExecutionHost   JSVNO JSV      S OS         Release    Hardware    Load   Cpu STT
--------------- ----- -------- - ---------- ---------- ---------- ----- ----- ---
[B]bhost         1000 LINKUP   - Linux      2.6.32-431 x86_64       0.2   0.2 ACT
host1              11 LINKDOWN - --         --         --             -     - INA
```

For the detailed information displayed by using qstat -Ef or qstat -Etf, [Baremetal] at the end of an ExecutionHost item indicates that bare metal server information is displayed. The resource amount of the bare metal server defined by the attach baremetal_host subcommand at registration can be referenced (Defined Baremetal Resources). The template that was used

to start the bare metal server can also be referenced (OpenStack Template).

[Indication example]

```
$ qstat -Ef
Execution Host: bhost [Baremetal]
        :
    Substitute Status = Normal
    OpenStack Template = Cent7_M
  Defined Baremetal Resources:
    Memory         = 8GB
    Number of Cpus = 2
    Number of GPUs = 1
  Resource Information:
    Memory         = Assign:     254976 Using:          - Maximum:     254976
    Swap           = Assign:     525312 Using:          0 Maximum:     525312
    Number of Cpus = Assign:          8 Using:          1 Maximum:          8
  GPU Information:
    Device[0]: GeForce 9800 GT
    TotalGlobalMem = 511 MB
        :
```

The job server information of the bare metal server can also be referenced by using qstat -S, -St, -Sf, or -Stf.

[Indication example]

```
$ qstat -S
JSVNO JobServerName    BatchServer      ExecutionHost    LINK BIND Queue     Jobs   Load
Cpu
----- --------------- --------------- --------------- ---- ---- -------- ---- ----- ---
--
 1000 JobServer1000   bsvhost          [B]bhost         UP       3 bq,bq2,*     0    0.0
0.0

$ qstat -Sf
Job Server Name: JobServer1000
    Job Server Number  = 1000
      :
    Execution Host = bhost [Baremetal]
      :
    Assign JobManipulator license = YES
    OpenStack Template = Cent7_M
  Defined Baremetal Resources:
    Memory         = 8GB
    Number of Cpus = 2
    Number of GPUs = 1
  Resource Information:
    Memory         = Assign:     254976 Using:          - Maximum:     254976
      :
```

(4) Binding and unbinding a bare metal server

A job server of the bare metal server registered by the attach baremetal_host subcommand can

be incorporated in a job operation by binding the job server to a queue. For more information about scheduling, see [JobManipulator].
By unbinding the relevant job server from the queue, it can be removed from a job operation.
The way to bind/unbind a bare metal server to/from a queue is the same as that for a usual execution host.

> [Notes]
> When binding a bare metal server to a queue, it is not possible to coexist that execution host in an environment on which to start a virtual machine and usual execution host in the queue.
> The queue to which to bind a bare metal server must be dedicated to a bare metal server.

(5) Resetting a bare metal server

If the started bare metal server entered LINKDOWN state due to failure or other causes, it is necessary to recover the bare metal server and then restore it to an operation (scheduling) by clearing the template information associated with the bare metal server on NQSV. (Otherwise, the bare metal server cannot be scheduled correctly.)
Use the reset baremetal_host subcommand of qmgr(1M) to clear the template information associated with the bare metal server. The administrator privilege is required to use this subcommand.

```
reset baremetal_host host = <host_name>
```

This subcommand can be used when the state of the JSV of the target bare metal server is LINKDOWN and the bare metal server has been started by any template (that is, a template name can be referenced as execution host information).
In addition, to use this subcommand, there must be no job stalled on the target host. If there is any stalled job, rerun or delete the relevant request.

## 15.3. Creating an OpenStack template

Information of an OS image and resources of a provisioning environment in conjunction with OpenStack is defined as an OpenStack template (hereafter referred to as a template).
A user can execute a job in the environment set in an OpenStack template by specifying the template for the --template option of a submit command (qsub(1), qlogin(1), or qrsh(1)).
The template contents are common to a virtual machine environment and bare metal environment.

### 15.3.1. Defining a template

A template is defined by a system administrator. Multiple templates can be defined. In a template, the following elements can be defined as information of a provisioning environment in conjunction with OpenStack.

| Element name | Definition |
|---|---|
| Template name | Template name.<br>A name can consist of up to 47 characters.<br>A space, double quotation mark ("), and @ symbol cannot be used in a template name.<br>Specify this template name when submitting a request. |
| OS image name (image) | Name of an OS disk image to be started by OpenStack.<br>This must be an image name that an execution host can use.<br>This can consist of up to 47 characters. |
| Flavor name (flavor) | Name of the OpenStack flavor.<br>This can consist of up to 47 characters. |
| Number of CPUs (cpu) | Number of CPUs to be assigned.<br>This must be an integer of 1 or larger.<br>This is also used as the limitation on the number of CPUs per job of the request for which the relevant template is specified. |
| Memory size (memsz) | Memory size to be assigned.<br>This must be an integer of 1 or larger followed by a unit (B, KB, MB, GB, TB, PB, EB).<br>This is also used as the limitation on the memory size per job of a request for which the relevant template is specified. |
| Number of GPUs (gpu) | Number of GPUs to be assigned.<br>This must be an integer of 0 or larger. Specify 0 if no GPU is assigned. The default is 0.<br>This is also used as the limitation on the number of GPUs per job of a request for which the relevant template is specified. |
| Estimated startup time (boot_timeout) | Timeout time of virtual machine and bare metal server startup scripts<br>This must be an integer of 1 to 2147483647. The unit is seconds. The default is 900 seconds.<br>This is also used by JobManipulator to assign a job to a bare metal server. |
| Estimated stop time (stop_timeout) | Timeout time of virtual machine and bare metal server stop scripts<br>This must be an integer of 1 to 2147483647. The unit is seconds. The default is 900 seconds. |
| Custom | If there is information uniquely defined as a startup environment, |

| | |
|---|---|
| definition<br>(custom) | describe it.<br>Up to 400 characters can be described. |
| Comment<br>(comment) | Comments for a template can be described.<br>Up to 255 characters can be described. |

[Notes]
When a request is transferred to another batch server by using a routing queue, use the same template configuration (the same template name and setting values) on all batch servers.

### 15.3.2. Using a template

(1) Creating a Create

Start qmgr(1M) with the administrator privilege and use the create openstack_template subcommand below to create a new template.

```
create openstack_template=<template_name> image=<OS_image> flavor=<flavor_name>
cpu=<cpunum> memsz=<memory_size> [gpu=<gpunum>] [boot_timeout=<timeout>]
[stop_timeout=<timeout>] [custom="<custom_define>"] [comment="<comment>"]
```

* For **cpu**, **memsz**, and **gpu** to specify the amount of each resource in a template, specify the same value set to **flavor** specifying the OpenStack flavor.

A template creation example is show below. The tables show OpenStack setting images and flavor settings.

Setting on OpenStack:

| Image name | Contents |
|---|---|
| rhel70 | RHEL7 has installed. |
| rhel71 | RHEL7.1 has installed. |

| Flavor name | Contents |
|---|---|
| small | Small-scale environment. CPU=1, Memory=1GB |
| medium | Medium-scale environment. CPU=2, Memory=2GB |
| large | Large-scale environment. CPU=4, Memory=4GB |

Templates defined by NQSV:

The following operation example defines four templates whose OpenStack configurations are small-scale (small) and medium-scale (medium) environments of RHEL7 (rhel70) and medium-scale (medium) and large-scale (large) environments of RHEL7.1 (rhel71).

```
$ /opt/nec/nqsv/bin/qmgr -Pm
```

```
Mgr: create openstack_template=os_70_small image=rhel70 cpu=1 memsz=1gb flavor=small
OpenStack Template os_70_small created.
Mgr: create openstack_template=os_70_medium image=rhel70 cpu=2 memsz=2gb flavor=medium
OpenStack Template os_70_medium created.
Mgr: create openstack_template=os_71_medium image=rhel71 cpu=2 memsz=2gb flavor=medium
OpenStack Template os_71_medium created.
Mgr: create openstack_template=os_71_large image=rhel71 cpu=4 memsz=4gb flavor=large
OpenStack Template os_71_large created.
```

(2) Deleting a Delete

Start qmgr(1M) with the administrator privilege and use the delete openstack_template subcommand below to delete the created template. However, if any request uses the target template, the template cannot be deleted.

```
delete openstack_template =<template_name>
```

(3) Editing a template

Start qmgr(1M) with the administrator privilege and use the set openstack_template subcommand below to edit the created template. However, if any request uses the target template, the template cannot be edited.

```
set openstack_template image=<OS_image>   <template_name>
set openstack_template cpu=<cpunum>   <template_name>
set openstack_template memsz=<memory_size>   <template_name>
set openstack_template gpu=<gpunum>   <template_name>
set openstack_template custom="<custom_define>"   <template_name>
set openstack_template comment="<comment>"   <template_name>
set openstack_template flavor=<flavor_name>   <template_name>
set openstack_template boot_timeout=<timeout>   <template_name>
set openstack_template stop_timeout=<timeout>   <template_name>
```

(4) Locking and unlocking a template

A template can be locked to temporarily prevent use when executing the above delete openstack_template and set openstack_template subcommands. If the locked template is specified for submitting a request, the submit operation will fail. There is no effect on the requests that have already been submitted.

For example, to edit a template, lock the target template to prohibit the template from being used when submitting a new request. If the request using the target template has already been submitted, wait until the request ends. When there is no request using the target template, edit it. After editing, unlock the template to make it usable.

Start qmgr(1M) with the administrator privilege and use the following subcommand to lock a

template.

```
lock openstack_template =<template_name>
```

Start qmgr(1M) with the administrator privilege and use the following subcommand to unlock a template.

```
unlock openstack_template =<template_name>
```

### 15.3.3. Displaying a template

Use qstat --template to reference the information of templates defined in a system.

[Indication example]

```
$qstat --template
[OpenStack Template]
==========================================================================================
==
Template   L Image       Flavor CPU  Memory GPU  Custom                Comment
---------- - ---------- ------ ---- ------ ---- -------------------- ------------------
--
os_70_smal - rhel70     small     1   1.0G    0 (none)               RHEL7 Small.
os_70_medi - rhel70     medium    2   2.0G    0 (none)               (none)
    :
```

Use qstat --template -f to reference the more detailed information of templates.

[Indication example]

```
$qstat --template -f
OpenStack Template: os_70_small
  Lock State   = UNLOCK
  OS Image     = rhel70
  Flavor       = small
  CPU Number   = 1
  Memory Size  = 1GB
  GPU Number   = 0
  Boot Timeout = 900
  Stop Timeout = 900
  Custom       = (none)
  Comment      = RHEL7 Small.
  Requests     = 0

OpenStack Template: os_70_medium
  Lock State   = UNLOCK
    :
```

### 15.3.4. Submitting a request with a template specified and locating a job

Use the --template option of qsub(1), qlogin(1), or qrsh(1) to submit a request with a template specified.

[Example]

```
$qsub --template= os_70_small -q bq -l elapstim_req=1000
```

When a request is submitted with the --template option specified, the number of CPUs, memory size, and number of GPUs that are defined in the specified template are used as the limits on resources (number of CPUs, memory size, number of GPUs) per job of the submitted request. These limit values are used to check the resources (number of CPUs, memory size, and number of GPUs per job) of a queue when the queue is submitted. The queue standard values are not applied.

When the request with a template specified uses virtual machines, one job is executed on one virtual machine.
When the request with a template specified uses bare metal servers, multiple jobs of the same request can be executed on one bare metal server; however, this configuration is not recommended.

# 16. Provisioning environment in conjunction with Docker

NQSV can execute a job on an isolated system (container) within an execution host in conjunction with Docker that can achieve container-based virtualization.
The function to define and manage a container image and resource information of this provisioning environment as a template is also provided.

## 16.1. Configuring a provisioning environment by using Docker

NQSV can dynamically configure a job execution environment in an execution host in conjunction with the Docker container creation function.
This section describes the conjunction between NQSV and the Docker container creation function using Docker of the following version as an example.

- Docker CE Version 19.03
- RedHat Enterprise Linux / CentOS 7.6, 7.7

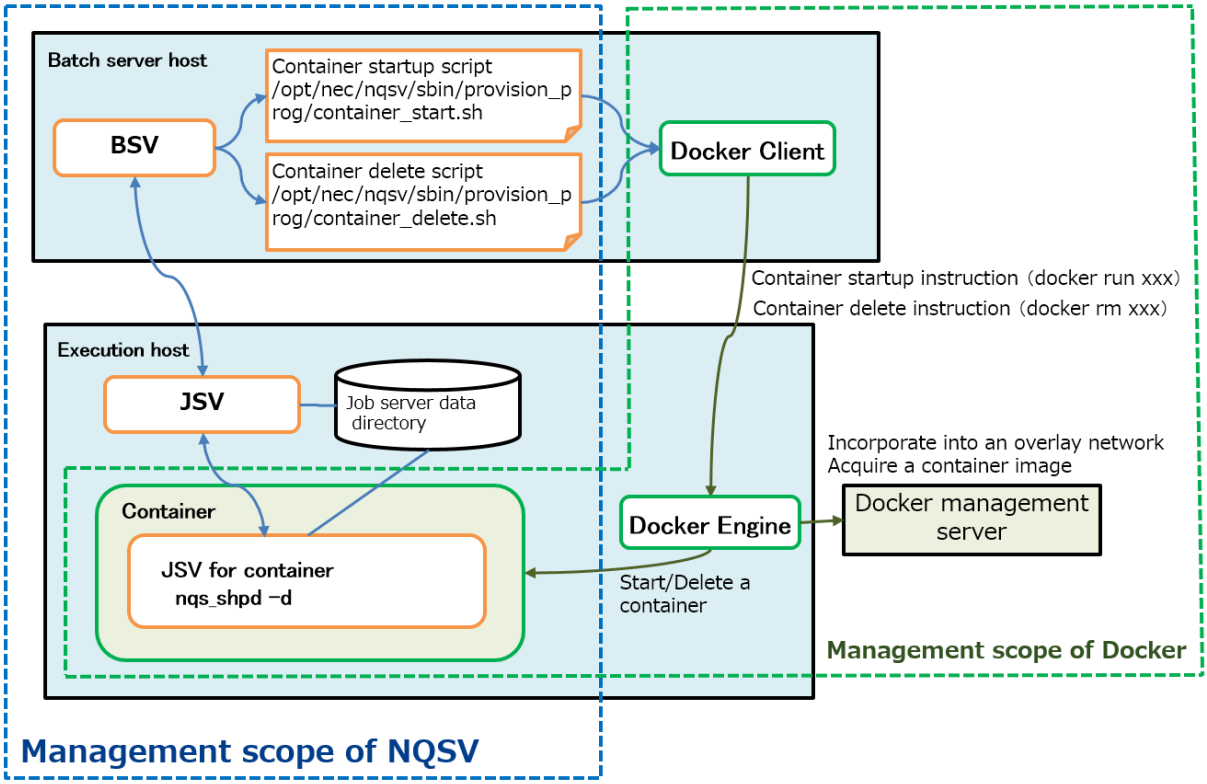



Figure 16-1 : Conceptual diagram of NQSV and Docker

The Docker management server is a machine on which Docker Registry a distributed KVS (Key-Value Store) can be used. Docker Registry is used to manage the container image repository. A distributed KVS (Key-Value Store) is used by Docker to build an overlay network.

An NQSV execution host is configured as a node on which to create a Docker container.
When a job starts, a batch server executes Docker Client by using a container startup script and instructs Docker Engine on the execution host to start a container by using a template (image and resources to start a container) including container job server information. Docker Engine acquires the instructed image from the Docker management server and incorporates the acquired image to an overlay network in conjunction with the Docker management server. Next, Docker Engine starts the container according to the resource information specified in the template and executes a job in the container.
When the job ends, the batch server executes a container delete script to instruct Docker Engine to delete the container.

### 16.1.1. Docker environment setting

To make Docker work together with NQSV, configure a Docker environment as follows in advance.

(1) Configuring a Docker environment on a batch server host

Configure an environment so that Docker Client can be used from a batch server host for Docker Engine.

(2) Configuring a Docker environment on an execution host

Configure an environment so that Docker Engine can be used on an execution host.

(3) Configuring a Docker management server

- Prepare a new Docker management server.
  (The configuration in which to use a batch server host as a Docker management server is also available.)
- Configure a Docker management server so that a distributed KVS required to use an overlay network as a network between containers can be used.
- Create an overlay network for jobs in advance.
  (It is also possible to create it dynamically in a container startup script.)
- Configure a Docker management server so that Docker Registry, repository of an images required to start a container, can be used.
- Create a container image and register it to Docker Registry in advance.

(4) Setting up a container boot image

Set up a container boot image to satisfy the following conditions:

- NQSV/JobServer package are installed in a container boot image. The boot image is

set up to start automatically in the **nqs_shpd -S** format when a container starts. It is not necessary to specify other nqs_shpd options (such as -h and -n).

- JSVDB (/var/opt/nec/nqsv/jsv) of an execution host is shared within a container.
- Docker can use an overlay network.
- Containers can communicate with each other (ssh, etc.) by using container host names.
- An account of a job execution user can be used in a container.
- A process in a container can communicate with an NQSV client host and execution host.

(*) Please edit the package version to the latest version when creating a Dockerfile based on this example.

The following shows a sample Dockerfile that creates an image to install SX-Aurora TSUBASA system software from yum repository on the internet, and set up to start a container automatically in the startup script(jsvstart.sh) which executes the nqs_shpd –d format and creates a job execution user.
In this sample, package files (TSUBASA-soft-release-2.0-1.norarch.rpm and MLNX_OFED_LINUX-4.7-1.0.0.1-rhel7.7-x86_64.tar) and config files (yum.conf, CentOS-Base.repo, TSUBASA-repo.repo, TSUBASA-restricted.repo) in the same directory as Dockerfile are placed.
[Dockerfile example]

```
FROM          docker.io/centos:7.7.1908
MAINTAINER    NEC
ADD           yum.conf /etc
ADD           CentOS-Base.repo /etc/yum.repos.d
ADD           TSUBASA-soft-release-2.0-1.noarch.rpm /tmp
ADD           TSUBASA-repo.repo /tmp
ADD           TSUBASA-restricted.repo /tmp
ARG           RELEASE_RPM=/tmp/TSUBASA-soft-release-2.0-1.noarch.rpm
RUN           yum -y install $RELEASE_RPM && \
              cp /tmp/*.repo /etc/yum.repos.d && \
              rm /tmp/*.repo && \
              yum clean all && \
              yum -y group install ve-container && \
              yum -y group install nec-sdk-runtime

RUN           yum -y group install nec-mpi-runtime nqsv-execution
```

```
RUN         yum -y install perl pciutils gtk2 atk cairo gcc-gfortran tcsh lsof libnl3 libmnl
ethtool tcl tk
ADD         MLNX_OFED_LINUX-4.7-1.0.0.1-rhel7.7-x86_64.tar /tmp
RUN         cd /tmp/MLNX_OFED_LINUX-4.7-1.0.0.1-rhel7.7-x86_64 && \
            ./mlnxofedinstall --user-space-only --without-fw-update -q --all
RUN         rm -rf /tmp/MLNX*

RUN         yum -y group install ve-container-infiniband

# Enable the next line if you use ScaTeFS
#RUN         yum -y group install scatefs-client-tsubasa-container

COPY        jsvstart.sh /tmp/jsvstart.sh
CMD         ["/tmp/jsvstart.sh"]
```

The following shows a sample script which creates a job execution user and group, executes NQSV/JobServer. Please place it in the same directory as Dockefile.

[jsvsgtart.sh example]

```
#!/bin/bash
/usr/sbin/groupadd -g ${NQS_CONTAINER_GID} ${NQS_CONTAINER_GNAME}
/usr/sbin/useradd -g ${NQS_CONTAINER_GID} -d ${NQS_CONTAINER_HOME} -u
${NQS_CONTAINER_UID} ${NQS_CONTAINER_UNAME}
/opt/nec/nqsv/sbin/nqs_shpd -d
```

### 16.1.2. Configuring a job server on an execution host

A job server that started in a container connects to a job server on an execution host. The job server in the container uses a host name acquired by the job server on the execution host as its own name at connection.

However, if host names that the job servers on the execution host and in the container can reference are different, expressly specify the host name that the job server in the container can reference by using the -H option when starting the job server on the execution host.

Option to start a job server on an execution host:

```
nqs_shpd -H <jhost> -h <bsv_host> -n <jsvno>
```

This enables to connect a job server in a container to a job server on an execution host by using

the host name <jhost> that can be referenced within the container.

### 16.1.3. Container Startup Script

A container is started by a shell script created by a system administrator. Locate a shell script in the following path.

Startup script: **/opt/nec/nqsv/sbin/provision_prog/container_start.sh**

Create a startup script to execute the following processes according to the NQSV and Docker execution environments.

(1) Processes to be executed in a container startup script

The following three processes must be executed in a container startup script.

1) **Starting a container**

Process to execute the docker run command according to the environment variables specified in the script to start a container.

2) **Waiting for containers to start**

Process to wait until all containers are fully started.

3) **Creating a container information file**

Process to describe the following items by delimiting with a space in the file specified by the environment variable NQS_CONTAINER_INFOPATH. Items of one job must be described in one line. (This is called a container ID file.)

- Started job ID
- Container host names (They must not be duplicated among jobs.)
- Container ID

[Example]

```
0 NQS-0-496-10 56fc8a257a34b92eaeae379bfdf34444693966e99f5f4e451eb11637a8b2a31e
1 NQS-1-496-10 47dd3f398c44a13cfddf451cb3345233724677a56f7f5b362ce32784b9f6b72a
```

When the above three processes are complete, the script terminates with **exit 0**.

If starting a container failed, for example, due to a failure of the docker run command during the processes **1)** to **3)** above, the relevant job number is described in the file specified by the environment variable NQS_CONTAINER_FAILINFOPATH. A job number of one job is described in one line. (This is called a FAIL file.)

Then the script terminates with **an exit code other than exit 0**.

(2) Environment variables to execute a container startup script

Specify the following environment variables to execute a container startup script.

| Environment variable | Value |
|---|---|
| NQS_CONTAINER_TEMPLATE_NAME | Template name |
| NQS_CONTAINER_IMAGE_NAME | Image name |
| NQS_CONTAINER_CPU | Number of CPUs |
| NQS_CONTAINER_GPU | Number of GPUs |
| NQS_CONTAINER_MEMORY | Memory size<br>(Example of "Size" and "Unit": 100 MB) |
| NQS_CONTAINER_CUSTOM | Custom definition |
| NQS_CONTAINER_QUE | Name of the queue to which a request was submitted |
| NQS_CONTAINER_RID | Request ID<br>Format: <seqno>.<mid><br>For <mid>, specify a number. |
| NQS_CONTAINER_INFOPATH | Path in which to save a container ID file |
| NQS_CONTAINER_FAILINFOPATH | Path in which to save a FAIL file |
| NQS_CONTAINER_PROCTYPE | "EXECUTION" (For forward processing)<br>"ROLLBACK" (For rollback) |
| NQS_CONTAINER_HOSTS | Host name and job number of an execution host on which to start a container<br>Format:<br><jobno>:<hostname>[ <jobno>:<hostname> ...]<br>* Use a space as a delimiter |
| NQS_CONTAINER_CPUSET_FUNC | Specify whether to enable or disable the NQSV CPUSET function.<br>"Disable" (disable)<br>"Enable" (enable) |
| NQS_CONTAINER_CPUSET_CPUS | CPU core number assigned to a job<br>(Only when the CPUSET function is enabled)<br>Format:<br><jobno>:<cpus>[ <jobno>:<cpus> ...]<br>* Use a space as a delimiter |
| NQS_CONTAINER_CPUSET_MEMS | Memory number assigned to a job<br>(Only when the CPUSET function is enabled)<br>Format:<br><jobno>:< mems>[ <jobno>:< mems> ...] |

| | |
|---|---|
| | * Use a space as a delimiter |
| NQS_CONTAINER_ASSIGNED_GPUS | GPU number assigned to a job<br>(Only when the GPU is required)<br>Format:<br><jobno>:< gpus>[ <jobno>:< gpus> ...]<br>* Use a space as a delimiter |
| NQS_CONTAINER_VE | Number of VEs |
| NQS_CONTAINER_ASSIGNED_VE_DEVS | VE number assigned to a job<br>(Only when the VE is required)<br>Format:<br><jobno>:< ves>[ <jobno>:< ves> ...]<br>* Use a space as a delimiter |
| NQS_CONTAINER_ASSIGNED_HCA_DEVS | The path of HCA on execution host.<br>Format:<br><jobno>:< path>[ <jobno>:< path> ...]<br>* Use a space as a delimiter |
| NQS_CONTAINER_UNAME | The job execution user name |
| NQS_CONTAINER_UID | The job execution user id |
| NQS_CONTAINER_GNAME | The job execution group name |
| NQS_CONTAINER_GID | The job execution group id |
| NQS_CONTAINER_WORKDIR | The job submission directory |
| NQS_CONTAINER_HOME | The home directory of job execution user |

(3) Rollback processing

If executing a container startup script failed (that is, the script ended with an exit code other than exit 0), the script will be executed again to perform a rollback processing. The environment variable NQS_CONTAINER_PROCTYPE is used to determine whether to execute a forward processing or rollback processing.

In a rollback processing, it is required to delete containers with container IDs described in the container ID file and wait for all of them to delete completely. NQSV executes the subsequent processes after the rollback processing assuming that all containers have been deleted. For the container that cannot be deleted in the rollback processing, the relevant job number must be described in a FAIL file.

(4) Timeout

Timeout monitoring is performed while a container startup script is executed. The value of the estimated startup time (Boot Timeout) set in a template is used as a timeout time.

In a forward processing of a container startup script, if the processing does not terminate even if the time set to Boot Timeout has passed, it is assumed that executing the startup

script failed. Then the container startup script is interrupted (by sending a KILL signal) and a rollback processing is performed.

In a rollback processing, timeout monitoring is also performed. If the processing does not terminate even if the time set to Boot Timeout has passed, executing the container startup script (rollback processing) is interrupted. However, if a rollback processing is interrupted due to a timeout, a container may remain on an execution host without deleting.

(5) Notes on creating a container startup script

When creating a container startup script to start multiple containers, select either of the following two operation types to be performed at detection of startup failure. The subsequent scheduling operation differs depending on which operation type is selected. Since there are advantages and disadvantages to both operation types, select either according to your operations.

1) **Waiting until the startup process of all containers are complete and detecting all the containers that could not start.**

This operation can detect all the containers that could not start among the containers to start. The advantage of this operation is that a schedule can be created to avoid the relevant host and assign a request to another host in the subsequent scheduling operation.

On the other hand, since it is required to wait until all target containers are fully started, it takes time to assign the request again after the detection of the first startup failure.

2) **Interrupting a process assuming that all startup processes failed if any container startup failure is detected.**

The advantage of this operation is that an action such as reassignment of a request can be performed immediately because a container startup failure can be detected as soon as its occurrence.

On the other hand, if another container cannot start after the detection of the first startup failure, that startup failure cannot be detected. Therefore, when the request is assigned to the relevant host next time, a startup failure may occur again.

### 16.1.4. Container delete script

A container is deleted by a shell script created by a system administrator. Locate a shell script in the following path.

Delete Script: **/opt/nec/nqsv/sbin/provision_prog/container_delete.sh**

Create a delete script to execute the following processes according to the NQSV and Docker execution environments.

(1) Processes to be executed in a container delete script

The following two processes must be executed in a container delete script.

1) **Deleting a container**

Process to execute the docker rm command according to the environment variables specified in the script to delete a container.

2) **Waiting for all containers to be deleted**

Process to wait until all containers are completely deleted (or until it is found that containers cannot be deleted).

* Even if a delete script fails, a rollback processing is not performed. Therefore, be sure to confirm that containers have been deleted completely. NQSV executes the subsequent processes assuming that all the containers have been deleted completely when the execution of the container delete script is complete.

When the above two processes are complete, the script terminates with **exit 0**.

If deleting a container failed, the relevant job number is described in the FAIL file specified by the environment variable NQS_CONTAINER_FAILINFOPATH. A job number of one job is described in one line. Then the script terminates with **an exit code other than exit 0**.

(2) Environment variables to execute a container delete script

Specify the following environment variables to execute a container delete script.

| Environment variable | Value |
|---|---|
| NQS_CONTAINER_TEMPLATE_NAME | Template name |
| NQS_CONTAINER_IMAGE_NAME | Image name |
| NQS_CONTAINER_CPU | Number of CPUs |
| NQS_CONTAINER_GPU | Number of GPUs |
| NQS_CONTAINER_MEMORY | Memory size<br>(Example of "Size" and "Unit": 100 MB) |
| NQS_CONTAINER_CUSTOM | Custom definition |
| NQS_CONTAINER_QUE | Name of the queue to which a request was submitted |
| NQS_CONTAINER_RID | Request ID<br>Format: <seqno>.<mid><br>For <mid>, specify a number. |
| NQS_CONTAINER_INFOPATH | Path in which to save a container ID file |
| NQS_CONTAINER_FAILINFOPATH | Path in which to save a FAIL file |

| | |
|---|---|
| NQS_CONTAINER_IDS | List of job numbers and container host names<br>Format:<br><jobno>:<container_hostname>[ <jobno>:<container_hostname> ...]<br>* Use a space as a delimiter |
| NQS_CONTAINER_PROCTYPE | "EXECUTION" |
| NQS_CONTAINER_HOSTS | Host name and job number of an execution host on which a container to delete exists<br>Format:<br><jobno>:<hostname>[ <jobno>:<hostname> ...]<br>* Use a space as a delimiter |

(3) Timeout

Timeout monitoring is performed while a container delete script is executed. The value of the estimated stop time (Stop Timeout) set in a template is used as a timeout time.

If the container delete script does not terminate even if the time set to Stop Timeout has passed, it is assumed that executing the script failed. Then the container delete script is interrupted (by sending a KILL signal).

If the container delete script is interrupted due to a timeout, a container may remain on an execution host without deleting.

> [Notes]
> If the container that was used to execute a job remains on an execution host because executing the container delete script failed, log in to the execution host and delete the relevant container directly.

### 16.1.5. Sample container startup and delete scripts

The sample container startup and delete scripts are installed in the batch server host. By referring these sample scripts, create a container startup and delete scripts including the above described processes and locate them in an appropriate path.

> [Notes]
> These sample scripts offer an implementation image of the minimum function. They do not guarantee operations on all Docker environments. When configuring an environment, implement appropriate processes according to your actual environment.

The summary of each script is described below.

(1) Sample container startup script

Installation path: /opt/nec/nqsv/sbin/provision_prog/container_start.sh.sample

Process overview:

When EXECUTION is set to the environment variable NQS_CONTAINER_PROCTYPE:

- The docker network command generates an overlay network dynamically.
- The docker run command starts a container on the execution host specified by the execution variable NQS_CONTAINER_HOSTS. Whether starting the container is successful is determined by the docker run command execution result.

When ROLLBACK is set to the environment variable NQS_CONTAINER_PROCTYPE:

- The docker rm command deletes the container with the container ID acquired from the container ID file specified by the environment variable NQS_CONTAINER_INFOPATH.
- Then, the docker network command deletes the dynamically generated overlay network.

(2) Sample container delete script

Installation path: /opt/nec/nqsv/sbin/provision_prog/container_delete.sh.sample

Process overview:

- The docker rm command deletes the container according to the container ID acquired from the container ID file specified by the environment variable NQS_CONTAINER_INFOPATH.
- Then, the docker network command deletes the dynamically generated overlay network.

#### 16.1.6. Notes on an execution host on which to start a container and queue

The queue to which to submit a request that starts a container and executes a job in the started container must be used only for starting a container.

Only the execution hosts on which Docker Engine can start a container must be bound to that queue. It is not possible to coexist with an execution host on which to start a virtual machine, bare metal server, and usual execution host.

### 16.2. Configuring a container template

Information of an OS image and resources of a provisioning environment in conjunction with Docker is defined as a container template (hereafter referred to as a template).

A user can execute a job in the environment set in a container template by specifying the template for the --template option of a submit command (qsub(1), qlogin(1), or qrsh(1)).

#### 16.2.1. Defining a template

A template is defined by a system administrator. Multiple template can be defined. In a template, the following elements can be defined as information of a provisioning environment in conjunction with Docker.

| Element name | Definition |
|---|---|
| Template name | Template name.<br>A name can consist of up to 47 characters.<br>A space, double quotation mark ("), and @ symbol cannot be used in a template name.<br>A user specifies this template name when submitting a request. |
| Image name (image) | Image name of a container to start.<br>This must be an image name that an execution host can use.<br>This can consist of up to 47 characters. |
| Number of CPUs (cpu) | Number of CPUs to be assigned.<br>This must be an integer of 1 or larger.<br>This is also used as the limitation on the number of CPUs per job of the request for which the relevant template is specified. |
| Memory size (memsz) | Memory size to be assigned.<br>This must be an integer of 1 or larger followed by a unit (B, KB, MB, GB, TB, PB, EB).<br>This is also used as the limitation on the memory size per job of the request for which the relevant template is specified. |
| Number of GPUs (gpu) | Number of GPUs to be assigned.<br>This must be an integer of 0 or larger. Specify 0 if no GPU is assigned. The default is 0.<br>This is also used as the limitation on the number of GPUs per job of the request for which the relevant template is specified. |
| Number of VEs (ve) | Number of VEs to be assigned.<br>This must be an integer of 0 or larger. Specify 0 if no VE is assigned. The default is 0.<br>This is also used as the limitation on the number of VEs per job of the request for which the relevant template is specified. |
| Number of HCAs (hca) | Number of HCAs which a job can use.<br>Format: (<for_io>, <for_mpi>, <for_all>)<br>Each value must be an integer of 0 or larger. Specify 0 if no HCA is used. The default is 0. |
| Estimated | Timeout time of a container startup script. |

| | |
|---|---|
| startup time (boot_timeout) | This must be an integer of 1 to 2147483647. The unit is seconds. The default is 900 seconds. |
| Estimated stop time (stop_timeout) | Timeout time of container stop script.<br>This must be an integer of 1 to 2147483647. The unit is seconds. The default is 900 seconds. |
| Custom definition (custom) | If there is information uniquely defined as a startup environment, describe it.<br>Up to 400 characters can be described. |
| Comment (comment) | Comments for a template can be described.<br>Up to 255 characters can be described. |

[Notes]

When a request is transferred to another batch server via a routing queue, use the same template configuration (the same template name and setting values) on all batch servers.

### 16.2.2. Using a template

(1) Creating a template

Start qmgr(1M) with the administrator privilege and use the create container_template subcommand below to create a new template.

```
create container_template=<template_name> image=<image> cpu=<cpunum>
memsz=<memory_size> [gpu=<gpunum>] [boot_timeout=<timeout>]
[stop_timeout=<timeout>] [custom="<custom_define>"] [comment="<comment>"]
```

An example to create a template named App_A with the following settings configured is shown below.

- Container image to start : App_A_Img
- Number of CPUs to be assigned : 2
- Memory to be assigned : 1GB
- Comment : For App_A

```
$ /opt/nec/nqsv/bin/qmgr -Pm
Mgr: create container_template=App_A image=App_A_Img cpu=2 memsz=1gb comment="For App_A"
Container Template App_A created.
```

(2) Deleting a template

Start qmgr(1M) with the administrator privilege and use the delete container_template subcommand below to delete the created template. However, if any request uses the target template, the template cannot be deleted.

```
delete container_template =<template_name>
```

(3) Editing a template

Start qmgr(1M) with the administrator privilege and use the set container_template subcommand below to edit the created template. However, if any request uses the target template, the template cannot be edited.

```
set container_template image=<image>   <template_name>
set container_template cpu=<cpunum>   <template_name>
set container_template memsz=<memory_size>   <template_name>
set container_template gpu=<gpunum>   <template_name>
set container_template custom="<custom_define>"   <template_name>
set container_template comment="<comment>"   <template_name>
set container_template boot_timeout=<timeout>   <template_name>
set container_template stop_timeout=<timeout>   <template_name>
```

(4) Locking and unlocking a template

A template can be locked to temporarily prevent use when the above delete container_template and set container_template subcommands are executed. If the locked template is specified for submitting a request, the submit operation will fail. There is no effect on the requests that have already been submitted.

For example, to edit a template, lock the target template to prohibit the template from being used when submitting a new request. If the request using the target template has already been submitted, wait until the request ends. When there is no request using the target template, edit it. After editing, unlock the template to make it usable.

Start qmgr(1M) with the administrator privilege and use the lock container_template subcommand below to lock a template.

```
lock container_template =<template_name>
```

Start qmgr(1M) with the administrator privilege and use the unlock container_template subcommand below to unlock a template.

```
unlock container_template =<template_name>
```

### 16.2.3. Displaying a template

Use qstat --template to reference the information of the template defined in a system.

[Display example]

```
$qstat --template
[Container Template]
================================================================================
```

```
Template   L Image      CPU  Memory GPU  Custom                Comment
---------- - ---------- ---- ------ ---- --------------------- --------------------
App_A      - App_A_Img     2   1.0G    0 (none)                For App_A
```

Use qstat --template -f to reference the more detailed information of templates.

[Display example]

```
$qstat --template -f
Container Template: App_A
  Lock State   = UNLOCK
  Image        = App_A_Img
  CPU Number   = 2
  Memory Size  = 1GB
  GPU Number   = 0
  Boot Timeout = 900
  Stop Timeout = 900
  Custom       = (none)
  Comment      = For App_A
  Requests     = 5
```

### 16.2.4. Submitting a request with a template specified and locating a job

Use the --template option of qsub(1), qlogin(1), or qrsh(1) to submit a request with a template specified.

[Example]

```
$qsub --template=App_A -q bq -l elapstim_req=1000
```

When a request is submitted with the --template option specified, the number of CPUs, memory size, and number of GPUs that are defined in the specified template are used as the limits on resources (number of CPUs, memory size, number of GPUs) per job of the submitted request. These limit values are used to check the resources (number of CPUs, memory size, and number of GPUs per job) of a queue when the queue is submitted. The queue standard values are not applied.

The request with a container template specified executes one job per container.

# 17. Custom Resource Function

The custom resource function is a function to control the custom resource amount that is used concurrently according to the defined custom resource information.

A system administrator defines a virtual resource. This virtual resource definition is called custom resource information. The custom resource information includes a custom resource name, resource consumption unit, control scope and upper limit of the custom resource amount that is used concurrently.
It is also able to define the following behavior of the custom resource. Collect the actual value of the custom resource or not. If it is set to collect, the kind of the collected value (moment or integrate) and the behavior of job termination (terminate or not) when the actual value exceed the limit value.

A user specifies a custom resource name and its usable amount for the --custom option of a job submit command (qsub(1), qlogin(1), qrsh(1)). A scheduler references the specified amount, sums the custom resource amount that is used concurrently, and controls scheduling so that the total resource usage does not exceed the defined custom resource upper limit. For more information about a scheduler, see [JobManipulator]. For more information about accounting and budge control of custom resources, see [Accounting & Budget Control].

Each queue has the default value and limit of the usage to set to a request. This is called custom resource usage limit information. When a batch server accepts a submitted request, the server determines whether to allow a job to submit according to this custom resource usage limit information. If necessary, the batch server applies the default value.

> [Notes]
> The custom resource function is available only for a batch request (local request) and interactive request. Therefore, the custom resource usage limit information can be set to these two queues.

If the custom resource is configured to collect the actual value, NQSV periodically collects the actual resource usage while the request is executing, and it could terminate the job if the actual value exceeds the limit according to the configuration.

## 17.1. Custom resource information

### 17.1.1. Custom resource information

Custom resource information is a virtual resource. Up to 20 custom resource information pieces

can be defined.

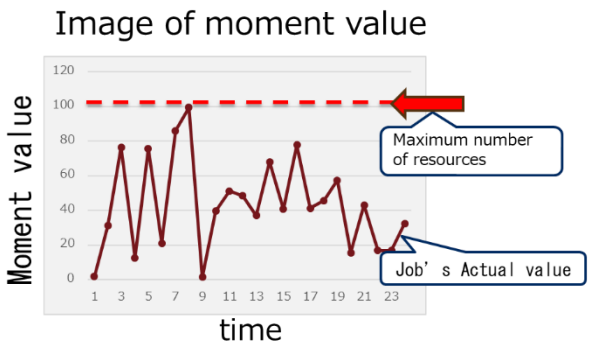
This custom resource information includes a resource consumption unit and the control scope and upper limit of the custom resource amount that is used concurrently.

The following table describes the details.

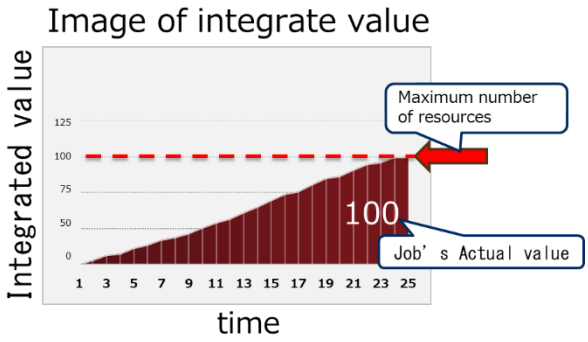
| Element | | explanation |
|---|---|---|
| Custom resource name | | Custom resource name<br>A name can consist of up to 15 characters.<br>A space, double quotation mark ("), and @ symbol cannot be used in a template name. |
| Consumption unit | | Unit to consume the custom resources specified when a request is submitted<br>Select either of the following:<br>・ **job** : job unit<br>・ **request** : request unit |
| Usage control information | Usage control target type | Type of the target whose custom resource usage is controlled.<br>・ **bsv**<br>Controls the custom resource usage of a BSV.<br>Set the maximum resource amount that can be used concurrently by the whole BSV and schedule jobs so that the resource usage does not exceed the specified resource amount.<br>・ **host**<br>Controls the custom resource usage of an execution host.<br>Set the maximum resource amount that can be used concurrently by one execution host and schedule jobs so that the resource usage does not exceed the specified resource amount.<br>host can be specified when the custom resource consumption unit is job. This type cannot be specified when the custom resource consumption unit is request. |
| | Usage control target type | Target whose usage is controlled.<br>Indicates the control scope of the maximum resource amount that can be used concurrently.<br>・ The usage control type is **bsv**:<br>The control scope is the entire BSV.<br>・ The usage control type is **host**:<br>The control scope is an execution host (default) or is |

| | | |
|---|---|---|
| | | specified individually by using an execution host name. |
| | Maximum number of resources | Maximum custom resource amount that can be used concurrently in the scope specified by the type and target described above.<br>This must be an integer of 0 to 2147483647. |
| Check mode | | Collect the actual value of the custom resource or not. If it is set to collect, the kind of the collected value.<br>**off**<br>Do not collect the actual value of the custom resource<br>**moment**<br>Collect the actual value and it is moment value.<br>**integrate**<br>Collect the actual value and it is integrated value. |
| Job termination | | The behavior of job termination. If it set to "on", the job is terminated when the actual value exceeds the limit value.<br>It could not be used when the Check mode is off. |
| Unit | | The unit of the custom resource. It could be specify 5 character. If the unit is not specified, the custom resource is treated as absolute number. NQSV not concern the conformity of the unit. (For example, the error not occurs when the actual value is moment and the unit indicates the integrate value.) |

For the consumption unit, be sure to select job or request.
The usage control information consists of the usage control target type, usage control target, and maximum resource amount that can be used concurrently. Multiple information pieces can be defined in one custom resource definition.
In case of check mode is moment or integrate, BSV handling collected value as following image.




In case of Check Mode is **moment**, BSV check that actual value at the time of collected is not over than maximum number of resources.




In case of check mode is **integrate**, BSV check that actual value which was integrated is not over than maximum number of resources.

### 17.1.2. Defining and deleting the custom resource information

Use the qmgr(1M) subcommands to define and delete the custom resource information. The administrator privilege is required to execute the subcommands.

The subcommands can be executed only when there is no request on a batch server. If a request exists, the subcommands to manipulate the custom resource information cannot be executed.

The details of how to manipulate the custom resource information using the qmgr subcommands are described below.

(1) Creating custom resource information

Use the "create custom_resource" subcommand to create custom resource information.

The following shows an example to create custom resource information whose name is Power and consumption unit is a job. In addition, the usage control target type is BSD and the maximum resource amount that can be used concurrently is 2000.

```
$ qmgr -Pm
Mgr: create custom_resource=Power consumer=job type=bsv available=2000
Custom_resoure Power created.
```

When creating a custom resource, be sure to select `job` or `request` for the consumption unit.

A custom resource can be created with omitting `type` and `available` that specify usage control information.

```
$ qmgr -Pm
Mgr: create custom_resource=License consumer=job
Custom_resoure License created.
```

The kind of actual value collection: check_mode, job termination when exceed the limit: terminate_job, unit of the custom resource: unit could be specified at the same time when

creating the custom resource. If there are not specified, the default value (check_mode=off, terminate_job=off, unit="") is applied.

```
$ qmgr -Pm
Mgr: create custom_resource=Power consumer=job type=bsv available=2000
check_mode=moment terminate_job=on unit=volt
Custom_resoure Power created.
```

Use the "edit custom_resource add" subcommand to add the usage control information. (For details, see "(6) Adding custom resource usage control information" below.)

> **[Notes]**
> For the custom resource that was created by omitting type and available specifications of usage control information, usage control using a scheduler is not performed.

(2) Changing the custom resource consumption unit

Use the "set custom_resource consumer" subcommand to change the custom resource consumption unit that was specified when creating custom resource information.

[Example]

```
$ qmgr -Pm
Mgr: set custom_resource consumer=request License
Set Consumer to Custom_resoure (License).
```

However, the unit cannot be changed from job to request if usage control information whose type is host has already been registered. In this case, delete the relevant usage control information before changing the unit.

(3) Changing the kind of actual value collection

Use the "set custom_resource check_mode" subcommand to change the setting of the kind of actual value collection when creating custom resource information.

[Example]

```
$ qmgr -Pm
Mgr: set custom_resource check_mode=moment Power
Set Consumer to Custom_resoure (Power).
```

However, check_mode could not be changed to "off" if the terminate_job is "on". If you want to change it to "off ", you must change the terminate_job to "off" first.

(4) Changing the setting of job termination

Use the "set custom_resource terminate_job" subcommand to change the setting of the job termination when the actual value exceed the limit when creating custom resource information.

[Example]

```
$ qmgr -Pm
Mgr: set custom_resource terminate_job=on Power
Set Consumer to Custom_resoure (Power).
```

However, terminate_job could not be changed to "on" if the check_mode is "off". If you want to change it to "on ", you must change the check_mode to "moment" or "integrate" first.

(5) Changing the unit

Use the "set custom_resource unit" subcommand to change the unit of the custom resource.

[Example]

```
$ qmgr -Pm
Mgr: set custom_resource unit=volt Power
Set Consumer to Custom_resoure (Power).
```

Equal or less than 5 characters could be specified for the unit.

(6) Add the amount control information to custom resource

Use the edit custom_resource add subcommand to add a custom resource usage control information piece. Specify the usage control target type (type) and target (target) and set the maximum resource amount that can be used concurrently (available). If they are already set, they will be overwritten.

The following shows an example to set the values for the custom resource information named "Power".

- The maximum resource amount that can be used concurrently of the entire BSV is 5000.
- The maximum resource amount that can be used concurrently per execution host is 100.
- The maximum resource amount that can be used concurrently of the execution host "host_a" is 120.

```
$ qmgr -Pm
Mgr: edit custom_resource add type=bsv available=5000 Power
Add Available_info from Custom_resource (Power).
Mgr: edit custom_resource add type=host available=100 Power
Add Available_info from Custom_resource (Power).
Mgr: edit custom_resource add type=host target=host_a available=120 Power
Add Available_info from Custom_resource (Power).
```

When type=bsv, the maximum resource amount that can be used concurrently of the entire BSV is set.

When type=host, the maximum resource amount that can be used concurrently per execution host is set. When type=host, the maximum resource amount that can be used concurrently can

be set to a certain execution host by specifying its name for target.

(7) Deleting the custom resource usage control information

Use the edit custom_resource delete subcommand to delete the custom resource information. Specify the usage control target type (type) and target (target) of the target information to delete. The following shows an example to delete information of the execution host host_a from the custom resource information named "Power".

```
$ qmgr -Pm
Mgr: edit custom_resource delete type=host target=host_a Power
Deleted Available_info from Custom_resource (Power).
```

When type=host, if target is not specified, whole information of the target execution host are deleted. (That is, the maximum resource amount that can be used concurrently per execution host and that of the execution host with the name specified are all deleted.)

(8) Deleting the custom resource information

Use the delete custom_resource subcommand to delete the custom resource information.

```
$ qmgr -Pm
Mgr: delete custom_resource=Power
Custom_resource Power deleted.
```

### 17.1.3. Displaying the custom resource information

Use the qstat(1) command with the --custom specified to display the custom resource information.

[Example]

```
$ qstat --custom
Custom Resource : Power
  Consumer = job
  Check Mode = Integrate
  Terminate Job = On
  Unit = MW
  Type = bsv :                          Available Resource Limit = 5000
  Type = host: Target = (default)       Available Resource Limit = 100
               Target = host_a          Available Resource Limit = 120

Custom Resource : License
  Consumer  = request
  Check Mode = Integrate
  Terminate Job = Off
  Unit = (none)
  Type = bsv :                          Available Resource Limit = 50

Custom Resource : Virtual
  Consumer = job
```

```
Check Mode = Moment
Terminate Job = Off
Unit = (none)
Type = bsv :                          Available Resource Limit = 100
Type = host: Target = (default)       Available Resource Limit = 1
```

## 17.2. Custom resource usage limit information of a queue

### 17.2.1. Custom resource usage limit information of a queue

When the custom resource information is defined, each queue has the custom resource usage limit information that is set to a request. This information includes the default usage, usage specification range, and setting of whether it is possible to specify the usage of a request to be uncontrolled. When a batch server accepts a submitted request, the server determines whether to allow a job to submit according to this custom resource usage limit information. If necessary, the batch server applies the default value.

The following table describes the details of the custom resource usage limit information of a queue.

| Element name | explanation |
|---|---|
| Custom resource name | Name of the custom resource name defined in the custom resource information. |
| Default usage set to a request | Default usage set to a request.<br>If no usage is specified when submitting a request, this default value is used.<br>The following can be specified.<br>・ Specify an integer of 1 to 2147483647.<br>・ **unused** (or, 0) < default value><br>* unused means that the specified custom resource will not be used (the usage is 0) and its usage will be uncontrolled. |
| Specification range of the usage set to a request | Upper and lower limits of the custom resource usage set to a request.<br>Specify an integer of 1 to 2147483647. |
| Whether it is possible to specify the usage of a request to be uncontrolled | Set whether it is possible to specify the usage of a request to be uncontrolled when the request is submitted.<br>・ yes Allows to set the usage of the submitted request to be uncontrolled (unused or 0). (Default)<br>・ no Does not allow to set the usage of the submitted request to be uncontrolled. |

### 17.2.2. Setting the custom resource usage limit information of a queue

Use the qmgr(1M) subcommands to set the custom resource usage limit information of a queue. The operator privilege is required to set the custom resource usage limit information of a queue. The following table shows the qmgr(1M) subcommands that are used to set items of the custom resource usage limit information of a queue and their default value.

| Setting item | qmgr(1M) sub-commands | Default |
|---|---|---|
| Default usage set to a request | set execution_queue custom_resource =*cr_name* standard=*std* *queue*<br>set interactive_queue custom_resource =*cr_name* standard=*std* *queue* | unused (0) |
| Specification range of the usage set to a request | set execution_queue custom_resource =*cr_name* range=(*min,max*) *queue*<br>set interactive_queue custom_resource =*cr_name* range=(*min,max*) *queue* | (1, 2147483847) |
| Whether it is possible to specify the usage of a request to be uncontrolled | set execution_queue custom_resource =*cr_name* permit_unused={ yes \| no } *queue*<br>set interactive_queue custom_resource =*cr_name* permit_unused={ yes \| no } *queue* | yes<br>* It is possible to specify the usage of a request to be uncontrolled |

The above three items of the custom resource usage limit information can be specified on one line. The following shows an example to set a custom resource information named "Power" with the following usage limit information for the batch queue bq: default value of 30, value range of (10,50), and for which setting the usage of a request to be uncontrolled is not allowed.

```
$ qmgr -Po
Mgr: set execution_queue custom_resource=Power standard=30 range=(10,50)
permit_unused=no bq
Set Custom_resource_info (Power). queue: bq
```

### 17.2.3. Displaying the custom resource usage limit information of a queue

Use the -Qf option of the qstat(1) command to display the custom resource usage limit information of a queue (Custom Resources).

[Example]

```
$ qstat -Qf
Execution Queue: bq@bsv
    Run State = Active
    Submit State = Enable
      :
  UserExit Script:
    (none)
  Custom Resources:
    Power            : Range (min,max) = 10,50        Std = 30          : Permit Unused =
No
    License          : Range (min,max) = 1,20         Std = unused      : Permit Unused =
Yes
```

```
    Virtual          : Range (min,max) = 1,1          Std = 1          : Permit Unused =
No
  Resources Limits:
    (Per-Req) Elapse Time Limit       = Max: UNLIMITED Warn: UNLIMITED Std:     3600S
    (Per-Job) CPU Time                = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
      :
  Kernel Parameter:
    Resource Sharing Group     = 0
    Nice Value                 = 0
      :
```

### 17.3. Requests when using the custom resource function

In an environment on which the custom resource function is used, a user specifies a custom resource name and its usable amount for the --custom option of a job submit command (qsub(1), qlogin(1), qrsh(1)). If no name and usable amount are set by a job submit command, the default custom resource names and usages set to a queue are applied.
A scheduler references the specified usable amount, sums the custom resource amount that is used concurrently, and controls scheduling so that the total resource usage does not exceed the defined custom resource upper limit.

For the custom resource usage of a request, set the environment variables of a job as follows:

```
NQSV_CR_<custom resource name>=<amount>
```

and

```
NQSII_CR_<custom resource name>=<amount>
```

Above 2 kinds of environment variable has same value.

[Environment variable setting example]

```
NQSV_CR_Power = 20
NQSV_CR_License = unused
NQSV_CR_Virtual = 1
```

The unit of the custom resource that request have, set the environment variables of a job as follows. There are 2 kinds of environment variable, but they have same value. If the unit is not specified for the custom resource, "" is set for <unit>.

```
NQSV_CR_UNIT_<custom resource name>=<unit>
```

and

```
NQSII_CR_UNIT_<custom resource name>=<unit>
```

These environment variables can be referenced in the following user defined process scripts.

- User EXIT script (for details, see 5.1.2. User EXIT)
- Hook script (for details, see 13. Hook Script Function)
- UserPP script (for details, see 14. User Pre-Post Script Function)

### 17.4. Resource monitoring script

If it is set the check mode to "moment" or "integrate", the resource usage on the execution host is monitored by resource monitoring script. Create the resource monitoring script for each custom resource, place it under /opt/nec/nqsv/sbin/custom_prog directory and set the script name to the same name as custom resouce name.
Following environment variables are applied when the script is executed.

| Environment variable | Value | Explanation |
|---|---|---|
| CR_ASSIGNED_VE | $0 \sim 2^{31}-1$ | Assigned VE number |
| CR_ASSIGNED_CORE | $0 \sim 2^{31}-1$ | Assigned CPU core number (*) |
| CR_NUMA_NODE | $0 \sim 2^{31}-1$ | Assigned NUMA node number (*) |
| CR_<custom resource name> | $0 \sim 2^{31}-1$ | The Limit of the custom resource |
| CR_UNIT_<custom resource name> | character | The Unit of the custom resource |
| CR_CPUNUM | $0 \sim 2^{31}-1$ | CPU number limit of the job |
| CR_GPUNUM | $0 \sim 2^{31}-1$ | GPU number limit of the job |
| CR_VENUM | $0 \sim 2^{31}-1$ | VE number limit of the job |
| CR_JOBID | <jobno>:<seqno>.<hosts> | Monitoring target Job ID |
| CR_TMPDIR | directory path | Temporary directory to save the information. |
| CR_EJID | 1~Max number of PID | Session ID of the job |

(*) It is only available when the socket scheduling feature is enabled.

Resource monitoring script is transferred to the execution host and it is executed periodically with root privilege. The monitored usage value is output to standard output. If the exit status of the script is not 0, it is treated as the monitoring failure and the job execution is interrupted.

The system manager must create the resource monitoring script to monitor the custom resource.

Two examples is shown below. One is the example which collects the I/O amount in VH, the other is the example which collects power consumption in VI.

[Script Example: Collection the I/O amount]

The I/O statics is recorded in '/proc/<pid>/io' file. The following script refers the file of job process, and gets the total amount(KiB) read and written of I/O. The script cannot get the I/O amount of Direct communication on VE.

```
#!/bin/bash
SAVED_DIR=${CR_TMPDIR}/ioacct
SID=${CR_EJID}
declare -A PID_HASH

if [ ! -e ${SAVED_DIR} ];then
    mkdir -p ${SAVED_DIR}
else
    for fl in `ls ${SAVED_DIR}`;
    do
        PID_HASH[${fl}]=`cat ${SAVED_DIR}/${fl}`
        rm ${SAVED_DIR}/${fl}
    done
fi

PIDS=`ps -s ${SID} -o pid`
IO=0
for pid in ${PIDS};
do
    IOFL="/proc/${pid}/io";
    if [ -e ${IOFL} ];then
        RCHAR=`cat ${IOFL} | grep rchar | awk -F ':' '{print $2}'`;
        WCHAR=`cat ${IOFL} | grep wchar | awk -F ':' '{print $2}'`;

        # B to KiB
        CHAR=$((${RCHAR}/1024+${WCHAR}/1024))
```

```
        IO=$(($${IO}+$${CHAR}))
        if [ -n "PID_HASH[$${pid}]" ];then
            PID_HASH[$${pid}]=0
            echo $${CHAR} > $${SAVED_DIR}/$${pid}
        fi
    fi
done
ACC=$${PID_HASH["ACC"]}
PID_HASH["ACC"]=0
for val in $${PID_HASH[@]};
do
    ACC=$(($${ACC}+$${val}))
done

echo $${ACC} > $${SAVED_DIR}/ACC
IO=$(($${IO}+$${ACC}))
echo $${IO}
exit 0
```

[Script Example: Collection power consumption]

In a rack with Intelligent PDU, you can get the power consumption with remote command through SNMP protocol. The following script example is that it get the power consumption from the PDU with snmapwalk command in the execution host.

Set the following variables if you use the script.

| Variable | : | Description |
|---|---|---|
| PDU_IP | : | Specify the ip address of PDU. |
| OUTLETS | : | Specify the outlet number of the execution host. |
| PWR_MIB | : | Specify Object ID which can get power consumption in the PDU MIB file. |

```
#!/bin/bash
SNMPBIN=/usr/bin/snmpwalk
PDU_IP=<IP_Address of PDU>
OUTLETS=<Outlet_numbers>
SUM=0
for outlet in $${OUTLETS};
do
    PWR_MIB=".1.3.6.1.4.1.13742.6.5.4.3.1.6.1.$${outlet}.6"
      POWER=`$${SNMPBIN} -v 2c -c public $${PDU_IP} $${PWR_MIB} | awk -F ':' '{print
```

```
  $4}'`
    if [ $? -ne 0 ]; then
        exit 1
    fi
    SUM=$(($${SUM} + $${POWER}))
done
echo $SUM
exit 0
```

# 18. Socket Scheduling

When using a NUMA architecture scalar machine (Linux) as execution host, NQSV can assign suitable resource (the number of CPU and memory) to the job (socket scheduling). It is also possible to split the host resources by cooperating with the CPUSET function of the Linux.

## 18.1. Socket Scheduling function

Socket scheduling function is the function to assign suitable resource (the number of CPU and memory) to the job. CPU is allocated by the core unit and memory is allocated by the socket unit.

The use of the socket scheduling feature can be enabled on a per queue configuration.

In a socket scheduling enabled queue, a core binding policy that controls how CPUs are assigned and a memory allocation policy that controls how memory are allocated in the job.

In addition, when socket scheduling is enabled on a queue, the following two features are available at the time of request submission

- The function to specify per job CPU number by using the number of socket
- The function to check the ratio of per job CPU number and per job memory size

### 18.1.1. Enabling socket scheduling function

To use the socket scheduling feature, run qmgr(1M) with operator privileges, and use the following sub-commands to enable the socket scheduling feature of the queue.

| Queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | set execution_queue numa_control = { on \| off } <queue> |
| Interactive queue | set interactive_queue numa_control = { on \| off } <queue> |

on　　Use socket scheduling function

off　　Don't use socket scheduling function

[Example]

```
$ qmgr -Po
Mgr: set execution_queue numa_control = on que1
Set Numa Control ON. queue: que1
```

[Notes]

- It is necessary to bind JSV which execution host is a NUMA architecture scalar machine (Linux) to the queue that socket scheduling feature is enabled.
- When binding more than one execution host to a queue, all execution hosts have the same socket configuration.
- When binding same execution host to multiple queues, the configuration for socket

scheduling feature must be same at all bound queues. It is not able to mix the ON and OFF configuration.

- When socket scheduling is enabled, the number of CPUs per logical host is automatically set to OMP_NUM_THREADS environment variable. On the job which uses VEs, set OMP_NUM_THREADS environment variable appropriately according to the number of VEs used in the job script.
- When the GPU-CPU Affinity feature is ON, the Socket Scheduling feature cannot be disabled. Please turn off the GPU-CPU Affinity feature before disable it. For details on the GPU-CPU Affinity feature, please refer to 18.3 GPU-CPU Affinity Feature.

### 18.1.2. Core bind policy

Following 2 kinds of policy for the socket assignment are available.

- **Socket concentration policy**
  Assign cores concentrated in a socket. Lesser free core in the socket is preferentially selected.
- **Socket distribution policy**
  Assign cores to be distributed among sockets. More free core in the socket is preferentially selected.

The core bind policy can be configured to each queue that enabled socket scheduling function. To configure it, use following qmgr (1M) sub-commands with operator privilege.

| Queue | qmgr(1M) sub-commands |
|---|---|
| Batch queue | set execution_queue numa_option core_bind_policy = <policy> <queue> |
| Interactive queue | set interactive_queue numa_option core_bind_policy = <policy> <queue> |

It is possible to specify following value as <policy>.

concentration : Socket concentration policy (default)
balance : Socket dispersion policy

### 18.1.3. Memory allocation policy

Following 3 kinds of policy for the memory allocation between the socket are available.

- **membind policy**
  Only the memory on the socket which the job's execution core belongs to is used. Swap is used when the memory insufficient.

- **localalloc policy**
  A memory on the socket which job's execution core belongs to is used with priority.

A memory on the socket that other jobs is using when the memory insufficient.

- **interleave policy**
  A memory of the socket assigned to a job is used alternately.
  If there is not enough memory, the memory on the socket of the other job is used.

Socket scheduling function can establish a memory allocation policy by the queue unit to the queue made more effective. qmgr (1M) is started by operator privilege in setting and the following sub-command is used.

| Queue | qmgr(1M) sub-commands |
|---|---|
| Batch queue | set execution_queue numa_option memory_allocation_policy = <policy> <queue> |
| Interactive queue | set interactive_queue numa_option memory_allocation_policy = <policy> <queue> |

It is possible to specify following value as <policy>.

| | |
|---|---|
| membind | membind policy |
| localalloc | localalloc policy (default) |
| interleave | interleave policy |

[Notes]

- If HugePages are configured on the system for SX-Aurora TSUBASA and the membind policy in socket scheduling is specified, there is a possibility that enough Hugepages are not available depending on the number of cpus requested by the job.
- VE programs on SX-Aurora TSUBASA require Hugepages to run fast. And it is recommended that the memory binding policy localalloc or interleave is specified.
- In case of the memory binding policy membind is used, please refer the SX-Aurora TSUBASA Installation Guide "4.11 HugePages Setting", and enable the MEMBIND option in the HugePages configuration command. It is necessary to bind JSV which execution host is a NUMA architecture scalar machine (Linux) to the queue that socket scheduling feature is enabled.

### 18.1.4. Specify per job CPU number limit by using number of socket

If socket scheduling is on, you can specify the number of sockets per job (socknum_job) instead of the number of CPUs per job (cpunum_job) in the qsub -l option when submitting a request.

(1) The way to specify the per job CPU number limit

Whether allow requests to be submitted by specifying the number of sockets per job or not, it can be set in a queue that the socket scheduling function is enabled.

To set this configuration, use qmgr(1M) command with the operator privileges and use the

following sub-commands.

| queue | qmgr(1M) sub-commands |
|---|---|
| Batch queue | set execution_queue submit_cpu_unit = { cpu \| socket \| any } <queue> |
| Interactive queue | set interactive_queue submit_cpu_unit = { cpu \| socket \| any } <queue> |

The meaning of the specify value is as follows in submit_cpu_unit.

cpu Only cpunum_job can be specified.

socket Only socknum_job can be specified.

any Either cpunum_job or socknum_job can be specified. (Default)

(2) Submitting request with socket number

you can specify the number of sockets per job (socknum_job) with the qsub -l option if the socket scheduling is ON and the queue is set to allow socknum_job to be specified.

To submit a request by specifying the number of sockets per job, use the -l socknum_job=<limit> option of qsub command. Note that socknum_job cannot be specified with the -l cpunum_job option.

[Example]

```
$ qsub -q bq -l socknum_job=4 job_script
Request 226.bsv.example.com submitted to queue: bq.
```

The number of sockets specified by socknum_job is automatically converted to the number of CPUs per job based on the sockets information on the hosts that bound to the queue.

In this automatic conversion, the number of CPUs is calculated as follows.

(The number of specified sockets) x (the number of CPUs per a socket (core) of the execution host that bound to a queue)

[Notes]

- This function is used to alternate specifying per job CPU number limit. It does not mean the CPU core always assigned by socket unit.
- Since this function uses information on the socket of the executing host, a submission error occurs if no JSVs are bound to the queue or all JSVs that bound to the queue have never been linked up.
- When all job servers bound to a queue and linked up are down, the number of CPUs is converted using the information before the down..

### 18.1.5. Check function of the ratio of per job CPU number and memory size

Check the ratio of the number of CPUs per job and memory size per job specified at qsub command are equal to the ratio of the number of CPUs (cores) and memory size on the socket of the execution host.
This function allows you to assign the CPUs to use the local memory in the socket is used as much as possible when a job is executed on the execution host.

This check the ratio of the number of CPUs and the amount of memory per job function can be used on the queue that the socket scheduling function is enabled. It can be configured to the queue by using following qmgr(1M) sub-command command with operator privileges.

| queue | qmgr(1M) sub-commands |
|---|---|
| Batch queue | set execution_queue numa_unit_check = { on \| off } <queue> |
| Interactive queue | set interactive_queue numa_unit_check = { on \| off } <queue> |

on ：a ratio checking function of CPU number and the memory size are used.
off ：a ratio checking function of CPU number and a ratio checking function of the memory size aren't used. (default)

[Notes]
- Since this function uses information on the socket of the executing host, a submission error occurs if no JSVs are bound to the queue or all JSVs that bound to the queue have never been linked up.
- When all job servers bound to a queue and linked up are down, the number of CPUs is converted using the information before the down..

### 18.1.6. Referring socket scheduling information

(1) Queue information

You can refer all setting about socket scheduling function (NUMA Control, NUMA option, Submit CPU Unit, NUMA Unit Check) in queue information that displayed in qstat - Qf.

[Example]

```
$ qstat -Qf
Execution Queue: que1@bsv1
    Run State    = Active
    Submit State = Enable
      :
    Restart option = {
      Ignore modify
    }
    NUMA Control = ON
    NUMA option = {
      Core Bind Policy = concentration
```

```
      Memory Allocation Policy = localalloc
    }
    Submit CPU Unit = any
    NUMA Unit Check = OFF
    Hold Privilege    = (none)
    Suspend Privilege = (none)
      :
```

(2) Execution host information

You can refer socket information and a socket usage information (Socket Resource Usage) of Linux execution host by using qstat - Ef. It is displayed when the socket scheduling function is enabled.

[Example]

```
$ qstat -Ef
Execution Host: host1
  Batch Server = bsv1
      :
  Average Information:
    LOAD (Latest  1 minute ): 0.230000
    LOAD (Latest  5 minutes): 0.270000
      :
  Cpuset Information:
    Resource Sharing Groups = {
      RSG Number 0  = Name: cpuset  Cpus: 0-31  Mems: 0-7
    }
  Socket Resource Usage:
    NUMA Nodes = {
      Node 0 (Cpus: 0-3)   = Cpu: 4/4 Memory: 0.5GB/4.0GB
      Node 1 (Cpus: 4-7)   = Cpu: 4/4 Memory: 0.5GB/4.0GB
      Node 2 (Cpus: 8-11)  = Cpu: 0/4 Memory: 0B/4.0GB
      Node 3 (Cpus: 12-15) = Cpu: 0/4 Memory: 0B/4.0GB
      Node 4 (Cpus: 16-19) = Cpu: 0/4 Memory: 0B/4.0GB
      Node 5 (Cpus: 20-23) = Cpu: 0/4 Memory: 0B/4.0GB
      Node 6 (Cpus: 24-27) = Cpu: 0/4 Memory: 0B/4.0GB
      Node 7 (Cpus: 28-31) = Cpu: 0/4 Memory: 0B/4.0GB
    }
```

(3) Job information

You can refer the assigned socket number that the running job by using in qstat - Jf when the socket scheduling function enabled. (Assigned Sockets)

[Example]

```
$ qstat -Jf
Request ID: 166.bsv1
    Batch Job Number = 0
    Execution Job ID = 3662
       :
    Remaining CPU Time = UNLIMITED
    Virtual Memory = 0.000000B
    Assigned Sockets = 0-1
```

:

## 18.2. CPUSET function

When using a scalar machine (Linux) supporting NUMA architecture as execution host, resource divide function equivalent to Resource Sharing Group (RSG) in SUPER-UX is offered in virtual way by using the CPUSET function of the Linux. This is called the CPUSET function. It is cooperate with the CPUSET function of the Linux OS. It divides the resources (CPU and memory) of execution host and make it relate to every queue, to schedule the huge resources efficiently.

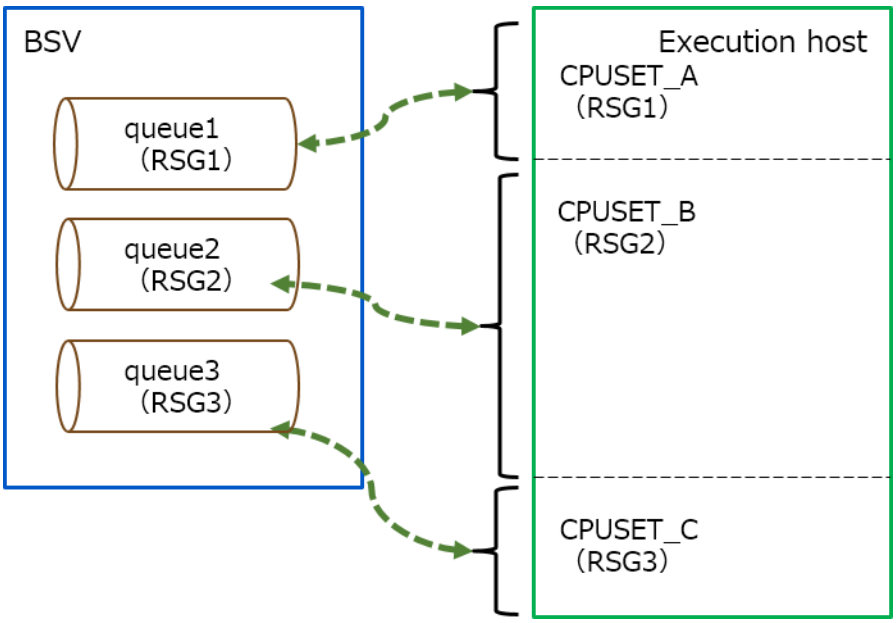



Figure 18-1 : Conceptual diagram of CPUSET function

Physical CPU core and memory node on each socket of execution host are grouped by the CPUSET function of the Linux and it is assigned to each job by the CPUSET function. The CPU core and the memory node assigned when the job execute by socket scheduling function are made in CPUSET which related to a queue (RSG) as CPUSET and a process of a job is executed in the CPUSET. A job can use the resources of the execution host exclusively by this.

Below is the key map which makes CPUSET for the jobs in a job of the request which was submitted in queue2 for which CPUSET_B on the execution host (RSG2) is used.

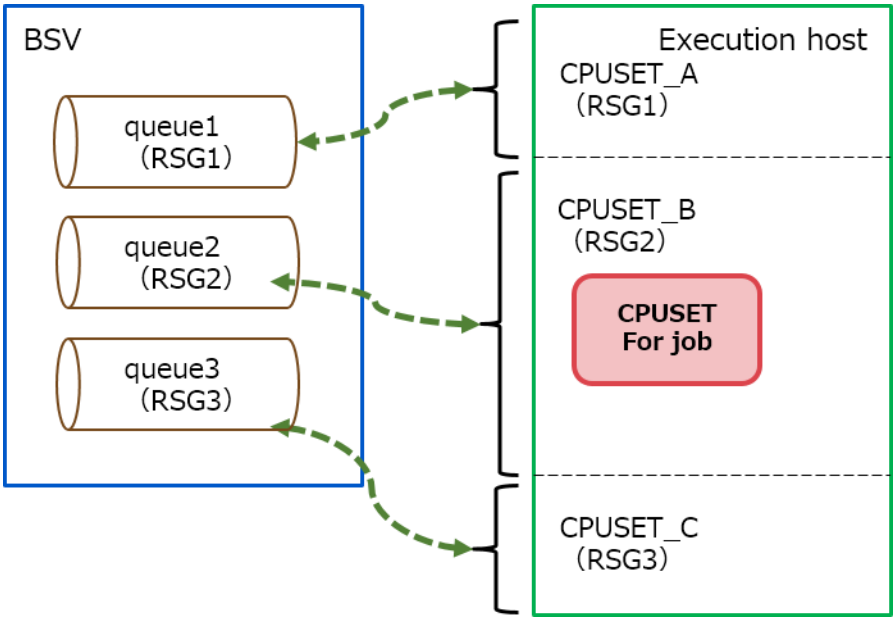



Figure 18-2 : Conceptual diagram of making CPUSET for jobs

CPUSET for jobs is made at the timing of execute starting. The job executes only using a CPU core and a memory node in the CPUSET(RSG) related to the queue the request submitted.

### 18.2.1. Configure CPUSET function

The CPUSET function is connected with socket scheduling function. Therefore please enables socket scheduling function of a queue first to use the CPUSET function. (Details are in 18.1.1. Enabling socket scheduling function.)

After enabling the function, please make the configuration file to define the CPUSET on the execution host for which the CPUSET function is used (cpuset.conf). Create cpuset.conf in /etc/opt/nec/nqsv of each execution host that describes by the following form.

- **The memory size of a socket (It is possible to omit.)**

```
Sizeof_MemoryNode <Memory size>
```

It is possible to omit. When omitting it, the memory size is acquired automatically from a host.

- **Definition of CPUSET (It is possible to define more than one. It is impossible to omit.)**

```
<CPUSET name>  <range of CPU core number>  <range of memory node number>  <RSG number>
```

Please divide into 1 line by a space character in 1 CPUSET and configure the following items.

- CPUSET name. It is the name of CPUSET generated on the Linux host.

· The range of the core number used in CPUSET (N-M) *

- The range of the memory node number used in CPUSET (N-M) *
- RSG number that correspond to CPUSET (0-31)

* If you want to describe non-consecutive values for the CPU core number and memory node number, separate the values by commas and do not include spaces. (N1,N2-M2,...)

[Notes]

- The first CPUSET must be defined to meet the following conditions
  - The name of the CPUSET must be "cpuset".
  - Specify the CPU core number and memory node number to be the amount of resources for the entire execution host.
  - The corresponding RSG number is "0".
  - The range of core numbers used in CPUSET is specified as per socket.
- In the following CPUSET definition lines, unique values are set for the CPUSET name and corresponding RSG number.
- If you change the setting of cpuset.conf, restart JSV.
- If you delete the CPUSET definition line that originally existed in cpuset.conf, please delete the CPUSET on the executing host manually.
- If you use the CPUSET function of NQSV, you must not manually create or delete the CPUSET on the executing host. (Except as described above)
- Please make sure that the CPU core number and memory node number do not overlap between each CPUSET. (except for "cpuset")
- Do not bind the same JSV to the queue using "cpuset" (RSG number 0) and other cpusets (RSG number 1 or above).

Below is a definition example of cpuset.conf.

```
Sizeof_MemoryNode 4.0GB
#######################################################
# A first CPUSET line is resources of the host total.
# The CPUSET sets 'cpuset'. The RSGNO must set '0'.
#######################################################
#CPUSET     CPUS        MEMS        RSGNO
cpuset      0-31        0-7         0
#######################################################
# Following CPUSET lines is divided resources.
# CPUSET and RSGNO should set unique values.
#######################################################
cpusetA     0-11        0-2         1
cpusetB     12-15       3           2
cpusetC     16-31       4-7         3
```

Set the RSG number of CPUSET to each queue after creating CPUSET. Use following qmgr

(1M) sub-command with operator privilege.

| queue | qmgr(1M) sub-command |
|---|---|
| Batch queue | set execution_queue kernel_param rsg_number = <value> <queue> |
| Interactive queue | set interactive_queue kernel_param rsg_number = <value> <queue> |

Below is the setting example when using CPUSET of RSG number 1 in que1.

```
$ qmgr -Po
Mgr: set execution_queue kernel_param rsg_number = 1 que1
Set RSG Number (Kernel-Parameter): que1
```

### 18.2.2. Referring CPUSET information

(1) Execution host information

You can refer the CPUSET information (Cpuset Information) of Linux execution host by using qstat -Ef command when the CPUSET function used.

[Example]

```
$ qstat -Ef
Execution Host: host1
  Batch Server = bsv1
      :
  Average Information:
    LOAD (Latest  1 minute ): 0.230000
    LOAD (Latest  5 minutes): 0.270000
      :
    CPU  (Latest 15 minutes): 0.008000
  Cpuset Information:
    Resource Sharing Groups = {
      RSG Number 0  = Name: cpuset  Cpus: 0-31  Mems: 0-7
      RSG Number 1  = Name: cpusetA Cpus: 0-11  Mems: 0-2
      RSG Number 2  = Name: cpusetB Cpus: 12-15 Mems: 3
      RSG Number 3  = Name: cpusetC Cpus: 16-31 Mems: 4-7
    }
  Socket Resource Usage:
    NUMA Nodes = {
      Node 0 (Cpus: 0-3)   = Cpu: 4/4 Memory: 0.5GB/4.0GB
      Node 1 (Cpus: 4-7)   = Cpu: 4/4 Memory: 0.5GB/4.0GB
      Node 2 (Cpus: 8-11)  = Cpu: 0/4 Memory: 0B/4.0GB
      Node 3 (Cpus: 12-15) = Cpu: 0/4 Memory: 0B/4.0GB
      Node 4 (Cpus: 16-19) = Cpu: 0/4 Memory: 0B/4.0GB
      Node 5 (Cpus: 20-23) = Cpu: 0/4 Memory: 0B/4.0GB
      Node 6 (Cpus: 24-27) = Cpu: 0/4 Memory: 0B/4.0GB
      Node 7 (Cpus: 28-31) = Cpu: 0/4 Memory: 0B/4.0GB
    }
```

(2) Queue information

You can refer the RSG number of CPUSET used by a queue in qstat -Qf (Kernel Parameter: Resource Sharing Group).

[Example]

```
$ qstat -Qf
Execution Queue: que1@bsv1
    Run State    = Active
    Submit State = Enable
      :
  Resources Limits:
    (Per-Req) Elapse Time Limit       = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
    (Per-Job) CPU Time                = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
      :
    (Per-Prc) Permanent File Capacity = Max: UNLIMITED Warn: UNLIMITED Std: UNLIMITED
  Kernel Parameter:
    Resource Sharing Group     = 1
    Nice Value                 = 0
      :
```

## 18.3. GPU-CPU Affinity function

The GPU-CPU Affinity function aims to improve the calculation speed of GPU jobs.
NQSV recognizes the physical distance between the GPU and CPU sockets on the execution host and allocates jobs to the CPU and GPU combinations that are in close proximity.

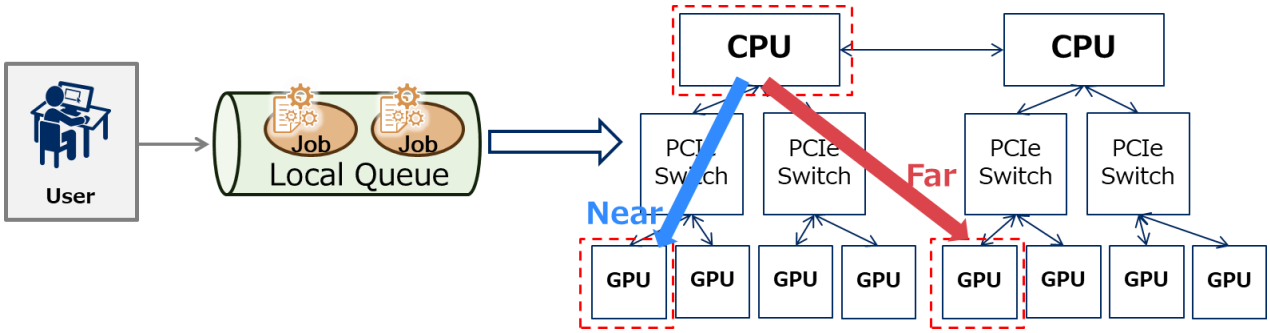



Figure 18-3 : Distance between GPU and CPU

The GPU-CPU Affinity function can be used on a queue which is the socket scheduling feature is enabled per queue.
When a request that uses the GPU is submitted to a queue which is the GPU-CPU Affinity feature enabled, a combination of CPU and GPU in close proximity can be assigned to the request.

### 18.3.1. Enable the GPU-CPU Affinity function

When use the GPU-CPU Affinity feature, execute qmgr(1M) command with operator privileges or higher, and use the following sub-command to set the GPU-CPU Affinity feature to ON of the queue.

| Queue | qmgr(1M) sub-command |
|---|---|
| Batch Queue | set execution_queue gpu_affinity = { on \| off } <queue> |
| Interactive Queue | set interactive_queue gpu_affinity = { on \| off } <queue> |

on : Use GPU-CPU Affinity feature
off : Don't use GPU-CPU Affinity feature (Default value)

[Example]

```
$ qmgr -Po
Mgr: set execution_queue gpu_affinity = on que1
Set GPU-CPU Affinity ON. queue: que1
```

[Notes]

- Please turn off the check function of the ratio of per job CPU number and memory size in the socket scheduling function. If this function is on, the GPU-CPU Affinity feature cannot be turned on. Also, when the GPU-CPU Affinity feature is on, the queue setting of check function of the ratio of per job CPU number and memory size cannot be changed to cpu or socket.
- When binding the same execution host to multiple queues, the GPU-CPU Affinity feature cannot be mixed on and off between those queues.
- Do not bind the same execution host to multiple queues with different settings for the number of CPUs per GPU. If bind, the GPU-CPU allocation will not be correct.

### 18.3.2. Number of CPUs per GPU

GPU-CPU Affinity feature allocates a combination of CPUs and GPUs in close proximity as a one bundle. Set the number of CPUs per GPU by the number of cores in a CPU socket divided by the number of GPUs that are close to that CPU socket.

This is the number of CPU cores per GPU, which allows select a combination of CPU and GPU that are in close proximity when scheduling jobs.

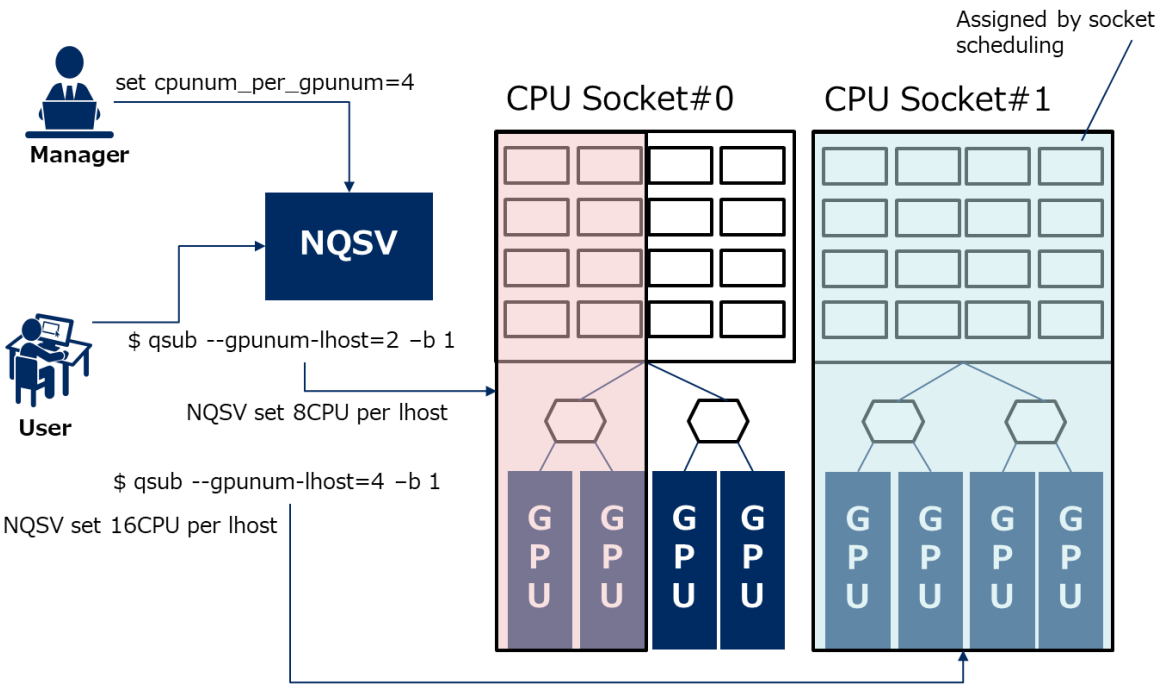



Figure 18-4 : Calculating the number of CPUs in a job by the number of CPUs per GPU

The number of CPUs per GPU should be calculated and set based on the following formula.

| Number of CPU cores in one socket / Number of GPUs physically close to the socket |
|---|

In the example above, the number of CPU cores in one socket is 16 and the number of GPUs physically close to the socket is 4, so the value of CPUs per GPU is 16 / 4 = 4.

The number of CPUs per GPU value can be set on a queues which is GPU-CPU Affinity turned on per queue. To set this, executing qmgr(1M) command with operator privileges and use the following sub-commands.

| Queue | qmgr(1M) sub-command |
|---|---|
| Batch Queue | set execution_queue cpunum_per_gpunum = <cpunum> <queue> |
| Interactive Queue | set interactive_queue cpunum_per_gpunum = <cpunum> <queue> |

<cpunum> can specify value from 0 to 2147483647. The default value is 1.

[Notes]

- If a value larger than the value calculated by the formula for calculating the number of CPUs per GPU is set, the number of cores to be allocated will be insufficient to allocate resources, and the job may fail with PRE-RUNNING. Therefore, be sure to check the number of GPUs and CPUs of the execution host, and set the appropriate number of CPU cores. If the settings are wrong and the job fails with PRE-RUNNING, please review the settings and then restart the job server where the job was executed.
- If there is a request in the queue, you cannot change the setting of this value.
- For queues where the GPU-CPU Affinity feature is turned on, the following restrictions are added when submitting a request.
  - It is not possible to submit requests with the --cpunum-lhost option or the -l cpunum_job option in the qsub(1)/qlogin(1)/qrsh(1) command. Also, when submitting hybrid requests, you cannot specify the --cpunum-lhost option or the -l cpunum_job option in all job groups.
  - When a request is submitted without specifying the --cpunum-lhost option or the -l cpunum_job option in the qsub(1)/qlogin(1)/qrsh(1) command, the number of CPUs per logical host is automatically calculated by the number of CPU per GPU, the standard value of the per-logical-host/per-job CPU limit of the queue will be ignored.
  - If the default value of the limit of the number of GPUs per logical host/per job for the batch queue and interactive queue is set to 0, be sure to specify a value of 1 or more for the --gpunum-lhost option or the -l gpunum_job option in the qsub(1)/qlogin(1)/qrsh(1) command.

- When the default value of the limit of the number of GPUs per logical host/per job for batch queue and interactive queue is set to 1 or more, you cannot specify a value of 0 for the --gpunum-lhost option or the -l gpunum_job option in the qsub(1)/qlogin(1)/qrsh(1) command.
- When submitting a hybrid request, be sure to specify a value of 1 or more in the --gpunum-lhost or -l gpunum_job option for all job groups.
- The qalter(1) command cannot be used to change the per-logical-host/per-job CPU and GPU limits for submitted requests.

・ When the CPU-CPU Affinity function is turned on in the destination queue of the routing queue, requests with a GPU number limit of 0 per logical host/per job or with the --cpunum-lhost (-l cpunum_job) option specified cannot be transferred.

### 18.3.3. Topology settings

The system administrator defines the CPU socket, socket number, and GPU topology information for PCIeSW in a file on the execution host. This file is called the device resource configuration file. This file is the same as the file described in [JobManipulator] 5.4.2 HCA and the Information of Topology, but the file format is different.

The settings cannot be changed during operation. If you want to change the settings, restart the JSV.

GPUs that are connected to the same CPU socket or the same PCIeSW (when the CPU socket and PCIeSW are connected) are grouped together and called a device group.

(1) Device group

Examples of device groups to which the GPU-CPU Affinity can be applied are as follows.

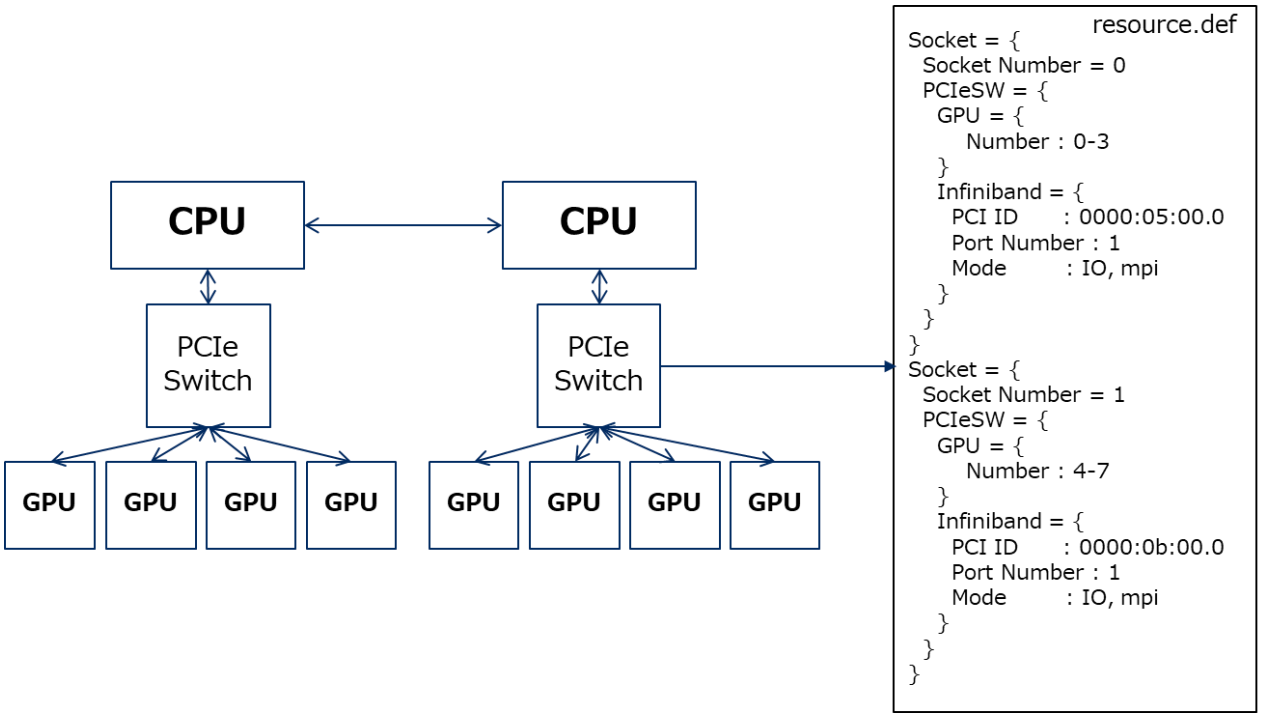



Figure 18-5 : GPU Topology Example

(2) Device resource configuration file

Device resource configuration file is /etc/opt/nec/nqsv/resource.def on the execution host.

(3) Device resource configuration file format

The format of the device resource definition file for the GPU-CPU Affinity function is as follows.

```
Format: <Resource>
        <Resource>: Resource information
Format : <Type> = { <List> }
        <Type>: Type of resource
Format : <Type> = Socket | PCIeSW | GPU
The meaning of each string is as follows.

● Socket : CPU socket

● PCIeSW:PCIeSW

● GPU: GPU

● Infiniband: HCA
```

```
<List> : A list of resource details. Topology information is expressed by describing
         resource information in a nested.
Format : <Resource> | <Attribute>
        <Attribute>: resource information detail
Format : <Name> : <Value>
        The detailed resource information that can be defined for each <Type> is as
follows.
        All of them must be set.

● Socket
   <Name> : <Value>
  Socket Number:socket number

● PCIeSW : PCI Switch
  No detailed resource information

● GPU
  <Name> : <Value>
  Number : GPU physical number (range can be specified)

● Infiniband
   <Name> : <Value>
  PCI ID  : PCI ID
  Port Number: port number
  Mode: Usage of HCA (IO and MPI can be specified. Multiple uses can be specified
  separated by commas.)
      IO  : Indicates I/O for direct communication.
      MPI : Indicates MPI for direct communication.
```

The character string to be set can be written in either upper or lower case.
If any of the following conditions are met, an error will occur when starting the JSV.
- PCIeSW was defined as a resource outside the Socket.
- GPU was defined as a resource outside of PCIeSW or Socket.
- A non-existent PCI ID was specified.
- VE" and "GPU" were specified at the same time, as described in [JobManipulator] 5.4.2 HCA and the Information of Topology.

(4) Device resource configuration file example

The following is an example of the configuration shown in Figure 18-5: GPU Topology Example, two PCIeSWs physically close to the sockets, and two GPUs installed in one PCIeSW.

```
    Socket = {
      Socket Number = 0
      PCIeSW = {
        GPU = {
          Number : 0-1
        }
        Infiniband = {
          PCI ID      : 0000:05:00.0
          Port Number : 1
          Mode        : IO, mpi
        }
      }
      PCIeSW = {
        GPU = {
          Number : 2-3
        }
        Infiniband = {
          PCI ID      : 0000:0b:00.0
          Port Number : 1
          Mode        : IO, mpi
        }
      }
    }
    Socket = {
      Socket Number = 1
      PCIeSW = {
        GPU = {
          Number : 4-5
        }
        Infiniband = {
          PCI ID      : 0000:13:00.0
          Port Number : 1
          Mode        : IO, mpi
        }
      }
```

```
    PCIeSW = {
      GPU = {
        Number : 6-7
      }
      Infiniband = {
        PCI ID      : 0000:19:00.0
        Port Number : 1
        Mode        : IO, mpi
      }
    }
  }
```

For the configuration without PCIeSW, the setting is without {} in PCIeSW as shown below.

```
  Socket = {
    Socket Number = 0
    GPU = {
      Number : 0-1
    }
  }
  Socket = {
    Socket Number = 1
    GPU = {
      Number : 2-3
    }
  }
```

(5) Refer to the device resource configuration file settings

You can check the settings of the device resource configuration file by using the qstat(1) -E -f command. The number next to PCIeSW indicates the ID of the device group.

```
  $ qstat -E -f
    ...
    Socket Resource Usage:
      NUMA Nodes = {
        Node 0 (Cpus: 0-3) = Cpu: -/4 Memory: -/7.9GB
        Node 1 (Cpus: 4-7) = Cpu: -/4 Memory: -/7.9GB
      }
```

```
Device Topology:
  Socket 0 = {
    PCIeSW 0 = {
      GPU: 0-1
      HCA: 0000:05:00.0 1 (IO,MPI)
    }
    PCIeSW 1 = {
      GPU: 2-3
      HCA: 0000:0b:00.0 1 (IO,MPI)
    }
  }
  Socket 1 = {
    PCIeSW 2 = {
      GPU: 4-5
      HCA: 0000:13:00.0 1 (IO,MPI)
    }
    PCIeSW 3 = {
      GPU: 6-7
      HCA: 0000:19:00.0 1 (IO,MPI)
    }
  }
```

If there is no PCIeSW, the PCIeSW part will not be displayed as shown below.

```
Device Topology:
    Socket 0 = {
        GPU: 0-1
    }
    Socket 1 = {
        GPU: 2-3
    }
```

If there is no device resource configuration file, the display will be as follows.

```
Device Topology:(none)
```

### 18.3.4. Using cgroups

The GPU-CPU Affinity feature allows you to limit the CPU and GPU allocated by cgroups. The restriction process is performed by shell scripts.

To restrict with cgroups, please place a shell script in the following path on the executing host. If this script does not exist, cgroups will not be restricted, but no error will occur. This script will be executed by root privilege on the executing host.

```
/opt/nec/nqsv/sbin/system_startup_prog/cgroups.sh
```

The following environment variables are passed when the shell script is executed.

| Environment variable name | Description |
|---|---|
| NQSV_CG_JOBID | Job ID. |
| NQSV_CG_GPUNUM | GPU number limit. |
| NQSV_CG_GPULIST | Assigned GPU number.<br>Example: If numbers 2 and 3 are assigned, the following values are obtained.<br>NQSV_CG_GPULIST=2,3 |
| NQSV_CG_CPUNUM | CPU number limit.<br>This is (the value of --gpunum-lhost) x (the number of CPUs per GPU set in the queue). |
| NQSV_CG_CPULIST | Assigned CPU number.<br>Example: If numbers 2 and 3 are assigned, the following values are obtained.<br>NQSV_CG_CPULIST=2,3 |
| NQSV_CG_MEMMAX | Maximum memory size limit.<br>The unit is represented by a single byte character at the end of the number. The concrete value is B, K, M, G, T, P, E.<br>B : Byte, K : KByte, M : MByte, G : GByte, T : TByte, P : PByte, E : EByte<br>In the case of unlimited, UNLIMITED is stored. |
| NQSV_CG_MEMWARN | Warning value for memory size limit.<br>The unit is represented by a single byte character at the end of the number. The concrete value is B, K, M, G, T, P, E.<br>B : Byte, K : KByte, M : MByte, G : GByte, T : TByte, P : PByte, E : EByte<br>In the case of unlimited, UNLIMITED is stored. |

When installing or updating NQSV/JobServer, a sample shell script is installed below, so

please refer to it to create cgroup.sh.

```
/opt/nec/nqsv/sbin/system_startup_prog/cgroups.sh.sample
```

The cgroup setting will be automatically deleted when the job is finished.

**[Notes]**

When cgroups are used to limit the CPUs and GPUs allocated by the GPU-CPU Affinity feature, the CUDA_VISIBLE_DEVICES environment variable will not be passed to the jobs because all GPUs visible to the job are those assigned to the job.

## 19. Failure Detection and Power Supply Control

NQSV provides the ability to detect execution host failures and power-saving capabilities by power control of the execution host. The node management agent described below, is required to use this function.

### 19.1. Failure Detection

In Linux, OSS (Open Source Software) for operation management such as Zabbix can be used to detect node abnormalities such as CPU high temperatures, H/W failures such as CPU failures, OS stalls, etc. (These are collectively called failures).
When the operations management OSS detects a failure, a node management agent that was started on the operations management host executes a failure notification command to notify the batch server of the failure.
In addition, if OSS for failure detection is not used, a simple failure detection script can be used to notify the batch server of the failure.

- It is possible to detect HW failure of a node in a short time, and exclude the node immediately.
- Failure type can be identified.

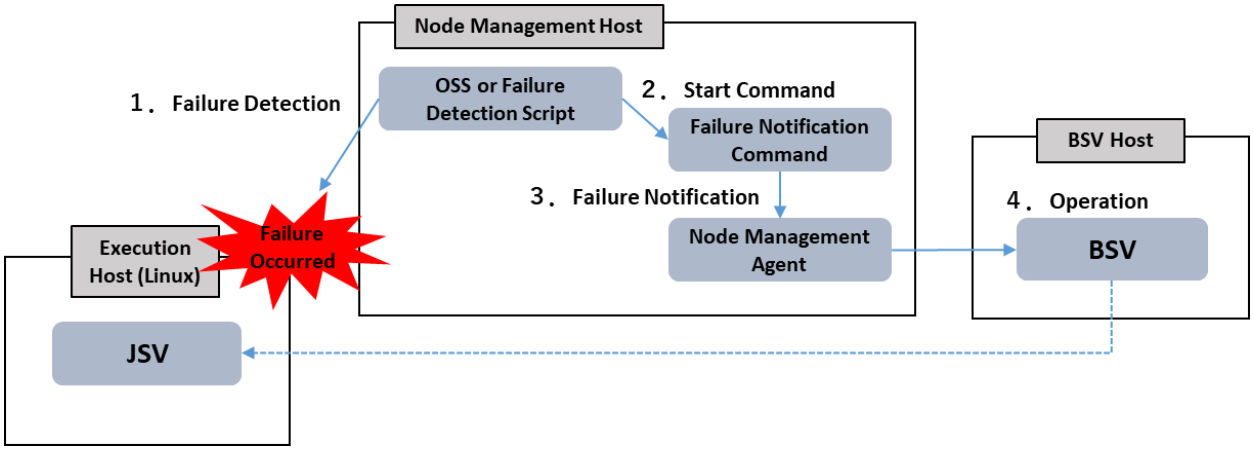



Figure 0-1 : Failure Detection

#### 19.1.1. Failure Detection Settings

(1) Failure detection with a simple failure detection script

On the operations management host, edit the simple failure detection script
/opt/nec/nqsv/sbin/ping_check.sh
and set the monitored execution host on the NODELIST= line.

```
#!/bin/bash
NODELIST="host1 host2 host3"

for node in $NODELIST;
do
    ping -c 5 $node >/dev/null 2>&1
    if [ $? -ne 0 ]; then
        /opt/nec/nqsv/sbin/nqs_ntfr -m "Down" $node >/dev/null 2>&1
    fi
done
```

On the operations management host, set this simple failure detection script to be executed periodically by cron. This allows the simple failure detection script to periodically check the execution hosts and notify the BSV of the failure if it cannot connect.
When a failure notification is received from a simple failure detection script, the BSV links down the JSV of the failed execution host.

It is possible to confirm the status of the execution hosts by qstat command with -Etf option. The following is a display image with the qstat -Etf command of the execution host that was notified of the failure by a simple failure detection script.

```
$ qstat -Etf host1
Execution Host: host1
    Batch Server = host1.example.com
    Current State           = Inactive
    State Transition Time   = Tue Sep 29 13:41:03 2017
    State Transition Reason = ABNORMAL STOP
    Message = Down
    Job Server Number  = 11
    LINK Batch Server = DOWN
```

(2) OSS failure Detection

When using OSS to detect faults, use the nqs_ntfr notification command. The failure notification command informs the BSV of the failure through the node management agent.

The failure notification command nqs_ntfr is installed on /opt/nec/nqsv/sbin.
The format is as follows:

```
nqs_ntfr [-o operation]  [-m message]  hostname | IP_address
```

The nqs_ntfr command informs the BSV that the execution host specified by the IP address or hostname has failed.
-o operation specifies how the BSV performs (do nothing/stop JSV/unbind JSV from queue) on the JSV in response to a failure notification.

It is possible to notify the details of failure with -m message. It is possible to confirm the contents of message by qstat command with -Etf option (Message item).

The details of options are as follows:

-o operation

Specifies the operation of the JSV during a failure notification.

One of the following can be specified for the operation.

Operate only when JSV is in LINKUP state.

① nothing To do nothing

② down To LINKDOWN JSV

③ unbind To unbind the JSV from the queue

If there is no -o option, it means "down".

The behavior of BSV by specifying –o operation, the LINK state of the JSV during failure notification, and the BIND state with the queue are as follows:

| operation | LINK state of JSV | | BSV behavior during failure notification |
|---|---|---|---|
| | LINK | BIND | |
| nothing | UP/DOWN | BIND /UNBIND | Do nothing to JSV.<br>BIND state of the JSV does not change. |
| down | UP | BIND /UNBIND | LINKDOWN JSV<br>The execution host state becomes INACTIVE.<br>BIND state of the JSV does not change |
| | DOWN | | Do nothing to JSV.<br>BIND state of the JSV does not change. |
| unbind | UP | BIND | UNBIND the JSV from the queue. |
| | | UNBIND | JDo nothing to JSV.<br>BIND state of the JSV does not change. |
| | DOWN | BIND /UNBIND | Do nothing to JSV.<br>BIND state of the JSV does not change. |

※ When other than the -o down is specified, BSV does not LINKDOWN JSV.

Linkdown is caused when the connection to the JSV is disconnected due to a failure.

-m message

Notify the message of the detail of failure

The maximum string length of message is 255 bytes.

If it exceeds 255 bytes, the excess string is truncated.

Displays with qstat -Etf, regardless of the status of the JSV.

Explains how to set failure detection using OSS(Zabbix). NQSV uses Zabbix version 2.4.6 to verify that Zabbix can use nqs_ntfr to notify node management agents of failures.

To detect a failure using Zabbix, the configuration steps are as follows: Please refer to Zabbix's

manual for detailed configuration operations.

Configure Zabbix to monitor the state of the execution host.

Create an Action and set the conditions for the failure to detect.

For example, if you want to detect memory exhaustion on the host as a failure, set it as Conditions as follows:

Trigger = host1: Lack of available memory on server host1

As an Operations for the created Action, please set the following items.

Target: Current host

Type: Custom script

Execute on: Zabbix server

Commands: /opt/nec/nqsv/sbin/nqs_ntfr -m "{TRIGGER.NAME}" {HOST.NAME1}

## 19.2. Power Supply Control

JobManipulator realizes a power saving function that links request scheduling and host start/stop.

In this function, the execution host is started using IPMI via a node management agent.

The node management agent uses ipmitool(/usr/bin/ipmitool) installed on the operations management host to access the BMC(Baseboard Management Controller) on the execution host and start the host.

To use the power control function, please make the following setting in advance.

- Set BMC to be available on each execution host
- Install the ipmitool on the node management host

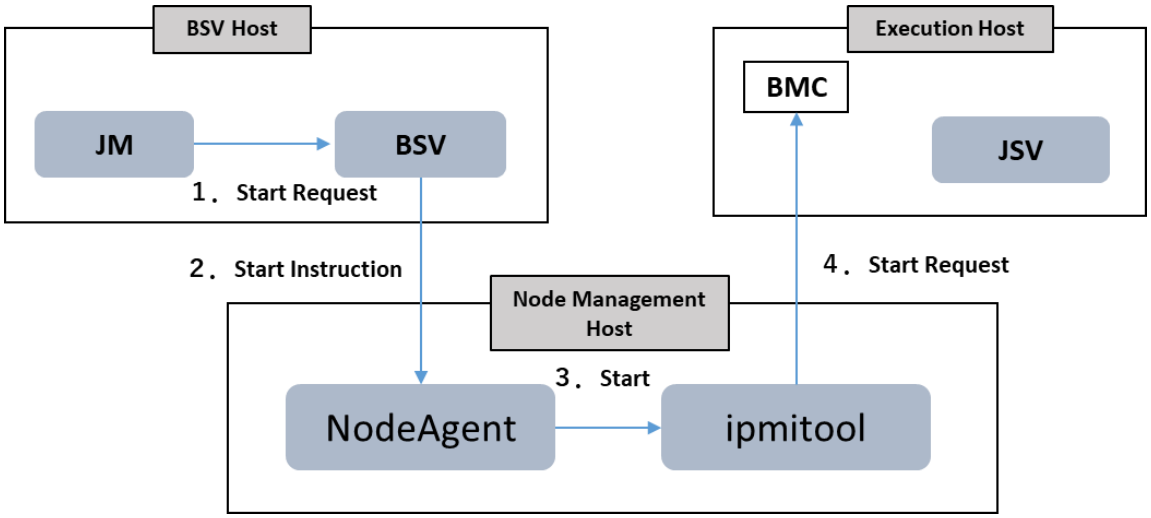


Figure 0-2 : Power Control Function

### 19.3. Node Management Agent Settings

In order to operate node management agent, the following settings are necessary.

1) Configure in order to use BMC on execution host.
   * If password to access BMC is set, a file in which the password is described is to be created.
     The file path is set at the step 3) and 4) below.
2) Install ipmitool in node management host
3) Configure for node management agent (/etc/opt/nec/nqsv/nag.conf)
4) Configure for managed host and access to BMC (/etc/opt/nec/nqsv/nag_nodelist)

In this section, 3) and 4) are explained.

◆ **/etc/opt/nec/nqsv/nag.conf (configuration file)**

In the configuration file for node management agent (/etc/opt/nec/nqsv/nag.conf), each item is set in the following format.

*Parameter* : *Value*

Lines preceded by # and blank lines are ignored as comment.

If same parameter are set on multiple lines, the last one is valid.

The followings can be set.

| *Parameter* | *Value*<br>(Settable value range or character number) | Description |
|---|---|---|
| BSVHOST | Character string<br>(Max:255 characters) | BSV host name. IP address may be set instead.<br>If BSVHOST is not set, default is localhost. |
| PORT | Integer<br>(1 to 65535) | BSV port number.<br>If PORT is not set, default is 602. |
| LOGFILE | Character string<br>(Max:1023 characters) | Pathname of Log file for node management agent.<br>If LOGFILE is not set, the default is /var/opt/nec/nqsv/node_agent_log. |
| LOGLEVEL | Integer<br>(0 to 4) | Output level of log.<br>The larger value is, the more detail information is output.<br>If LOGLEVEL is not set, the default is 1.<br>Definition of each level is as follows.<br>0: Only warning and error information without debug information are output.<br>1: In addition to warning and error information, node management agent start/stop, batch server connection, and failure detection information are output. |

| | | |
|---|---|---|
| | | 2: In addition to 1, debug information is output.<br>3: In addition to 2, information of processing of all packets is output.<br>4: In addition to 3, information of contents of all packets is output. |
| LOGSIZE | Integer<br>(1 to 2147483647)<br>or UNLIMITED | The largest size of one log file (unit:byte)<br>The maximum value is 2147483647 (2GB-1).<br>If LOGSIZE is not set or UNLIMITED is set, the maximum value is 2147483647. |
| LOGSAVE | Integer<br>(0 to 2147483647) | The number of files to retain when a logfile exceeds the specified size in LOGSIZE.<br>When the number of save file exceeds the number specified in LOGSAVE, the oldest file will be deleted.<br>If 0 is set, no log files are retained.<br>If LOGSAVE is not set, the default is 3. |
| IPMIUSER | Character string<br>(Max:47 characters) | User name to access BMC<br>On executing ipmitool, set user name specified with -U option.<br>If IPMIUSER is not set, -U option should not be specified. |
| IPMIPASSWDFL | Character string<br>(Max:1023 characters) | Pathname of the file in which password is described to access BMC.<br>On executing ipmitool, set the path of password file specified with -f option.<br>If IPMIPASSWDFL is not set, -f option should not be specified. |

In case that the above setting change is made while node management agent is in operation, the setting change needs to be reflected by using the following systemctl command for node management agent.

```
root# systemctl reload nqs_nag.service
```

◆ **/etc/opt/nec/nqsv/nag_nodelist (node list file)**

In nag_nodelist, target execution host of node management agent, IP address of BMC, and pathname of file containing username and password to access BMC are to be set.

Only if both user name and password different from IPMIUSER and IPMIPASSWDFL in nag.conf need to be used on each execution host, please set pathname of file containing user name and password.

If the settings exist in both nag_nodelist and nag.conf, settings in nag_nodelist is given priority.

Each setting can be set in the following format with space separator.

```
<host name>|<IP address>    <BMC's IPaddress> [<user name> <password file path>]
```

Lines preceded by # and blank lines are ignored as comment.

If same parameter are set on multiple lines, the last one is valid.

Multiple node management agents can be operated. Do not define duplicate execution hosts in each agent's node list file.

If node file list is modified, node management agent needs to be restarted.

```
root# systemctl restart nqs-nag.service
```

### 19.4. Failure Detection Function Settings

(1) Failure Detection by Simplified Failure Detection Script

Please set target execution host to be monitored to NODELIST= line by editing the simplified failure detection script, /opt/nec/nqsv/sbin/ping_check.sh on node management host.

```
#!/bin/bash
NODELIST="host1 host2 host3"

for node in $NODELIST;
do
    ping -c 5 $node >/dev/null 2>&1
    if [ $? -ne 0 ]; then
        /opt/nec/nqsv/sbin/nqs_ntfr -m "Down" $node >/dev/null 2>&1
    fi
done
```

In addition, on node management host, cron needs to to be configured so that this simplified failure detection script is periodically executed by cron. This enables simplified failure detection script to periodically check whether execution host is alive and to notify BSV of failure if it cannot connect with execution host.

BSV which receives failure notification from simplified failure detection script brings JSV of execution host with failure in LINKDOWN.

State of execution host can be checked with qstat -Etf.
The followings are display image of qstat -Etf for execution host about which simplified failure detection script notified failure.

```
$ qstat -Etf host1
Execution Host: host1
    Batch Server = host1.example.com
    Current State           = Inactive
    State Transition Time   = Tue Sep 29 13:41:03 2017
    State Transition Reason = ABNORMAL STOP
    Message = Down
    Job Server Number  = 11
    LINK Batch Server = DOWN
```

(2) Failure Detection by OSS

Failure notification command, nqs_ntfr is used in order to detect failure by using OSS. Failure notification command notifies BSV of failures via node management agent.

Failure notification command, nqs_ntfr is installed in /opt/nec/nqsv/sbin.
Its format is as follows.

```
nqs_ntfr [-o operation]  [-m message]  hostname | IP_address
```

nqs_ntfr command notifies BSV of failures about execution host specified by *hostname* or *IP_address.*
Action (no action, JSV down and unbind) taken by BSV against JSV on receiving failure notification is specified by -o *operation*. In addition, failure detail can be notified by specifying -m *message*. Content of *message* can be checked with qstat -Etf.

Details of each options are as follows.

- -o *operation*

  Specify action against JSV on failure notification.
  *operation* can be set to either of the followings.
  Action is taken only if JSV is in LINKUP state.
  - nothing  no action taken against JSV
  - down  bring JSV into LINKDOWN
  - unbind  UNBIND JSV from queue

  If -o option is not specified, the default action is "down".

  BSV action depending on combination of -o *operation,* JSV LINK state and BIND state with queue is as follows.

| *operation* | JSV state on failure notification | | BSV action on failure notification |
|---|---|---|---|
| | LINK | BIND | |
| nothing | UP/DOWN | BIND /UNBIND | No action against JSV.<br>BIND state of JSV does not change. |
| down | UP | BIND /UNBIND | Bring JSV into LINKDOWN.<br>State of execution host turns INACTIVE.<br>BIND state of JSV does not change. |
| | DOWN | | No action against JSV.<br>BIND state of JSV does not change. |
| unbind | UP | BIND | UNBIND JSV from queue. |

| | | | |
|---|---|---|---|
| | | UNBIND | No action against JSV.<br>BIND state of JSV does not change. |
| | DOWN | BIND<br>/UNBIND | No action against JSV.<br>BIND state of JSV does not change. |

***** Unless -o down is specified, BSV does not take action to bring JSV into LINKDOWN.
If connection with JSV is lost due to failure, its state turns LINKDOWN.

- - m *message*
  Failure detail is specified with *message*.
  The maximum length of *message* is 255 byte.
  If character string exceeds 255 byte, excess characters are discarded.
  Any JSV state can be checked with qstat -Etf.

As for OSS settings to execute failure notification command on failure detection, settings for Zabbix are as follows.
For NQSV, Zabbix Version2.4.6 was used and it was confirmed that failure could be notified node management agent of by using nsq_ntfr from Zabbix.
Setting procedure to notify failure using Zabbix are as follows. For detail setting operations, please refer to Zabbix manual.

- Configure Zabbix so that state of execution host can be monitored.
- Create "Action" and specify condition to detect failures
  For example, in order to detect lack of memory on host as failure, the followings need to be specified as "Conditions".

  | |
  |---|
  | Trigger = host1: Lack of available memory on server host1 |

- As Operations of created action, the followings are to be specified.

  | |
  |---|
  | Target: Current host |
  | Type: Custom script |
  | Execute on: Zabbix server |
  | Commands: /opt/nec/nqsv/sbin/nqs_ntfr -m "{TRIGGER.NAME}" {HOST.NAME1} |

## 19.5. Node Health Check Function

NQSV can use shell scripts to check whether the node on which a job is executed is healthy when the job starts (PRE-RUNNING) and when the job finishes (POST-RUNNING). NQSV can detect various HW failures such as CPU, GPU, VE, HCA, etc. by creating shell scripts for the check according to the operation policy of each site. NQSV will re-run the request that triggered the check when a node health check detects a failure, and can UNBIND or LINKDOWN the problem node to remove it from operation. In addition, e-mail notification can be sent to the owner of the request using a pre-defined user notification script.

### 19.5.1. Overview of Node Health Check Settings

To use the node health check function, configure the settings in the following order.

1. Create and place health check scripts
2. Configuring actions on failure detection nodes
3. Setting up scripts for user notification
4. Adjusting health check time with Elapse margin

The following sections describe the detailed settings for using the node health check function and the operation when a failure is detected.

### 19.5.2. Health Check Scripts

The health check script is to be created by the system administrator according to the operational policy of each site. The script name and location is the following path on the execution host where the JSV is running.

/opt/nec/nqsv/sbin/healthchk.sh

If the above script does not exist or if the execution permissions are not set properly, no node health check will be performed. To perform a node health check on the execution host, please create this script in advance and place it on the execution host.

The following environment variables are set by NQSV when the health check script is executed. The script uses this information to check the status of various HW.

| Environment Variables | Description | Value |
|---|---|---|

| | | |
|---|---|---|
| NQSV_HEALTHCHK_TIMING | Timing of Health Check | PRE-RUNNING<br>POST-RUNNING |
| NQSV_HEALTHCHK_RID | Request ID | Normal request:<br><sequence number>.<hostname><br>Parametric request:<br><sequence number>[].<hostname> |
| NQSV_HEALTHCHK_JID | Job ID | <job number>:<request ID> |
| NQSV_HEALTHCHK_SID | Session ID of the job | 1 to maximum value of process ID |
| NQSV_HEALTHCHK_REQOWNER | Owner name of the request | User name |
| NQSV_HEALTHCHK_REQOWNERGRP | Group name of the owner of the request | Group name |
| NQSV_HEALTHCHK_CPULIST | List of CPU numbers assigned to the job. The value is set only when the socket scheduling feature is enabled. | CPU number, separated by commas.<br>Example: 0,1,2,3,4 |
| NQSV_HEALTHCHK_VELIST | List of VE node numbers assigned to the job. The value is set only if VE is used for the job. | VE node number, separated by commas.<br>Example: 0,1,2,3,4 |
| NQSV_HEALTHCHK_GPULIST | List of GPU numbers assigned to the job. The value is set only for jobs that use a GPU. | GPU number, separated by commas.<br>Example: 0,1,2,3,4 |
| NQSV_HEALTHCHK_HCALIST | List of HCA device names assigned to the job. The value is set only for jobs that use HCA. | A HCA device name string separated by commas.<br>Example: 0000:3e:00.0, 0000:4d:00.0 |

To avoid taking wrong actions against requests and nodes due to excessive failure detection, please perform necessary failure detection based on the information of resources allocated to the job in the script. For example, when detecting VE failures, check the value of the environment variable NQSV_HEALTHCHK_VELIST, and if there is a VE node number, take the appropriate action. Note that since CPU resources are always used by jobs, failure detection can always be performed.

The result of the health check should be returned as the exit code of the shell script, and NQSV will acquire the exit code and check whether there was a failure in the health check. If the exit code of the shell script is 0, it assumes that there is no failure and continues to execute the request. If any other value is returned, it is determined that there is a failure and the request is re-executed or action is taken on the node concerned.

### (1) Sample Script for VE and HCA Health Check

The health check sample script for VE and HCA failure detection is installed on the execution host. Please refer to this sample to create a script that implements the necessary failure detection process, and place it in the appropriate path.

> Note: This sample is provided to show a minimum implementation image of the functions, and does not guarantee operation in all environments. When building the environment, please implement the appropriate process according to the actual environment.

Installation path:

/opt/nec/nqsv/sbin/healthchk.sh.sample

Processing Overview:

In this sample, failure detection of VE and HCA is performed in the order of VE to HCA. For VE, we use the VE node abnormality check command provided by VEOS (/opt/nec/ve/veos/libexec/ve_check_job: called ve_check_job) to check for abnormalities in VE and VEOS at the start and end of a job. For HCA, NQSV calls the shell script for HCA failure detection (/opt/nec/nqsv/sbin/hcachk.sh: called hcachk.sh) that is installed on the execution host to check for failures in the HCA device.

If an failure is detected in the VE failure detection, the HCA failure detection is not performed. When a failure is detected, information about the job, failure code, detection timing, and other information is output to /var/log/messages of the Linux OS. Note that if no VE or HCA is assigned to the job, the relevant detection process is not performed.

VE fault detection is performed in the following flow.

1. Checks if a VE has been assigned to the job. If it is not assigned, skip the check process.
2. Checks for the existence of ve_check_job. If it does not exist or does not have the permission to execute, output the log and skip the VE failure detection as no failure.
3. Call ve_check_job with the options (-s: at the start of the job, -e: at the end of the job) indicating the check timing, VE node number, and session ID of the job according to the health check timing (PRE-RUNNING or POST-RUNNING).
4. If the return value of ve_check_job is 0, the system assumes there is no failure. Otherwise, it judges that there is a failure. When a failure is detected, the related information is output to /var/log/messages of the Linux OS as warning information. When multiple VEs are assigned to a job, even if a failure is detected in one of the VEs during the process, the failure detection process is continued for the remaining VEs.
5. If a failure is detected in one or more of the VEs assigned to the job, it returns exit code 1 as the result of the health check.

HCA failure detection is performed in the following flow.

1. Checks if HCA is assigned to the job. If it is not assigned, skip the check process.
2. Checks for the existence of hcachk.sh. If it does not exist or does not have the permission to execute, it will output a log and skip HCA failure detection as no failure.
3. Call hcachk.sh with the HCA device name. The timing of the health check is not specified.
4. If the return value of hcachk.sh is 0, it assumes there is no failure. Otherwise, it is determined that there is a failure. When a failure is detected, the related information is output to /var/log/messages of the Linux OS as warning information. When multiple HCAs are assigned to a job, if a failure of one HCA is detected during the process, the process is aborted without continuing the failure detection process for the remaining HCAs.
5. If a failure is detected in one of the HCAs assigned to the job, exit code 2 is returned as the result of the health check.

### 19.5.3. Setting the action of the failure detection node

When a failure is detected by the node health check, one of the following actions can be selected per execution host for the failed node: unbind/down/nothing.

When "unbind" is selected, all the queues bound to the node where the failure was detected will be unbound. Except for the job that triggered the failure detection, the jobs running on the node will continue to run. Select "unbind" when the failure affects only the job in question and does not affect other jobs running on the same node when a failure occurs in VE or GPU.

When "down" is selected, the node where the failure was detected will be LINKDOWN. The jobs running on the node will be stalled, except for the job that triggered the failure detection. When the forced re-run function of the running job of JobManipulator is ON, the job is forced to be re-run. Jobs that are assigned to the back of the node will be unassigned and reassigned to other nodes. If the failure affects all jobs running on the node, select down.

If you select "nothing", no action will be taken on the detected failed node.

The following subcommands of qmgr(1M) are used to configure the settings. Operator privileges are required for the setting.

| Subcommand of qmgr(1M) |
| --- |
| `set job_server health_check_action = <action> job_server_id=<jsvid>` |
| `set job_server health_check_action = <action> job_server_id=(<jsvids>)` |

The following is an example of configuring UNBIND for the number 1 of the JSV.

```
$ qmgr -Po
Mgr: set job_server health_check_action = unbind job_server_id=1
Set node health check action.
```

The result of the configuration is displayed as follows with the -S -f option of the qstat(1) command.

```
$ qstat -Qf
Job Server Name: JobServer0035
    Job Server Number  = 1
      :
    HCA Failure Check = OFF
    Health Check Action = UNBIND
:
```

### 19.5.4. Configure the script for user notification

When the **-m a** option is specified at the time of the request submission, if a failure is detected by the node health check at the time of the job end, the owner of the request who detected the failure is notified by e-mail. If the failure is detected at the start of the job, the user will not be notified because the job has not yet started and will be rolled back to the QUEUED state and

re-executed.

The following is the format of the notification email.

| |
|---|
| NQSV request: <rid> aborted.<br>Request name: <Name of job script><br>Request owner: <owner><br>Mail sent at: <date and time of sending><br>[jobno <job number>]: Job has aborted by node health check error. |

If you also specify a mail address for notification with the **-M** option when submitting a job, e-mail notification is sent to the specified mail address. If the **-m a** option is not specified at the time of job submission, no e-mail notification is sent. By default, the following shell script on the batch host will perform mail notification.

/opt/nec/nqsv/sbin/nofity_prog/nqsv_notify.sh

By modifying the above shell script, you can customize the means of notification, the contents of the notification, and the destination of the notification. Also, if you have a site-specific notification script, you can specify the path to the script in the batch server configuration file. For details, please refer to 2.3.18 User Notification Script Settings.

(1) Script for user notification

User notification scripts can be created by system administrators according to the operational policies of each site. You can also decide the script name and location. The created script should be applied to the batch server configuration file by specifying the path.

The following environment variables are set by NQSV when the user notification script is executed.

| Environment variable | Description | Value |
|---|---|---|
| NQSNOTIFY_RID | Request ID | Normal request:<br><sequence number>. <hostname><br>Parametric request:<br><sequence number>[].<hostname> |
| NQSNOTIFY_TITLE | Subject of the email | NQSV request:<request ID> aborted. |
| NQSNOTIFY_CONTENTS | Body of the email | Request name: <job script name><br>Request owner: <owner> |

| | | |
|---|---|---|
| | | Mail sent at: <date sent><br>[jobno <job number>]: Job has aborted by node health check error. |
| NQSNOFITY_REQOWNER_EMAIL | Email address of the request's owner | Email address |
| NQSNOTIFY_REQOWNERGRP | Group name of the request's owner | Group Name |
| NQSNOTIFY_QUEUE | Queue name to submit the request | Queue Name |

### 19.5.5. Adjusting Health Check Time with Elapse Margin

The health check process is executed by PRE-RUNNING and POST-RUNNING of the request. If the node health check takes a long time, the PRE-RUNNING and POST-RUNNING times will be longer. You can tune this time by setting the Elapse margin in JobManipulator appropriately. This will allow you to operate without overlapping resource occupation time with requests that are assigned backwards. For more information about Elapse Margin, please refer to 3.1.7 Elapse Margin in JobManipulator chapter.

### 19.5.6. Rerun the fault detection request

If the node health check detects a fault, rerun the request that triggers the failure detection.
If a failure is detected at the start of the job (PRE-RUNNING), the request is returned from the PRE-RUNNING state to the QUEUED state and rerun. If there are multiple jobs in a request, if the health check fails for only some jobs, the health check returns to the QUEUED state after the health check at the end of the job for the job with a successful health check.
If only some jobs fail the health check, the health check at the end of the jobs with a successful health check is passed back to the QUEUED state.
If a fault is detected at the end of the job (POST-RUNNING), the request is returned to the QUEUED state and rerun.
The reason for the state transition of the request is "SYSTEM_FAILURE".

### 19.5.7. Accounting and budget of failure detection requests

If the node health check detects a failure, the node health check fails flag to the account of the request that triggers the failure detection. It is able to check the fault information for each job or request using the --hw-failure option in the account reference commands scacctjob(1)/scacctreq(1). If a fault is detected, 20 is displayed in the HW FAILURE column as

follows. If a request with more than one job detects one or more failures in the jobs, set a fault detection flag for the request. The display image of the account reference command is as follows:

```
$ scacctjob --hw-failure -I 1

==========================================================================================

====

JOB                          REQUEST  … QUEUED   START    END                 CPU    REAL      HW

ID     REQUEST_ID             NAME    …  NAME     TIME     TIME     TIME      (SECS)  (SECS) FAILURE

==========================================================================================

====

32816 1.bsv1(0000)             job1   …   bq       20:29:48 20:29:48 20:32:48     0.02  179.75      20

34769 1.bsv1(0001)             job1   …   bq       20:29:48 20:29:48 20:32:49     0.08  180.12       0
```

```
$ scacctreq --hw-failure -I 1

==============================================================================================

REQUEST                 REQUEST  USER      QUEUE    QUEUED   START    END          CPU    REAL            HW

ID                      NAME     NAME      NAME     TIME     TIME     TIME      (SECS)  (SECS) STATUS  FAILURE

==============================================================================================

1.bsv1                  job1     user      bq       20:29:48 20:29:48 20:32:49     0.02     180 DELETED      20
```

It is not charged for requests that trigger fault detection. In the case of parametric requests, it is not charged for sub-requests that detect faults, but it is charged for sub-requests that can be executed successfully.

## 20. Failover

### 20.1. Redundancy Function without using EXPRESSCLUSTER

Redundancy function can realize simple redundancy of Batch server, Accounting server, or Scheduler without using EXPRESSCLUSTER.

Please refer "20.2.Redundancy Function using EXPRESSCLUSTER" if you want to use EXPRESSCLUSTER for redundancy.

This redundancy function monitors failures using OSS such as Zabbix or the accompanying simple failure monitoring program. When Redundancy function detects a failure, primary server to preconfigured stand-by server.

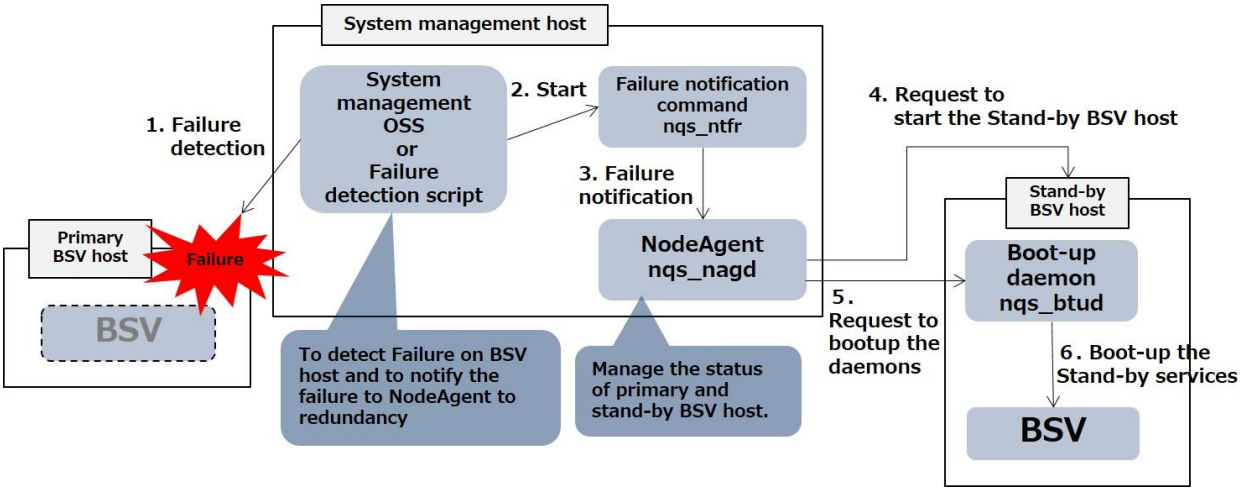



Figure 20-1 : The whole image of Redundancy function

On this manual mainly explains about redundancy of batch server. But you can duplicate the accounting server and scheduler too, by using same way.

### 20.1.1. Install Boot-up Daemon

When Redundancy function is used, NQS/ResourceManager package needs to be installed on the stand-by host.

After install, Boot-up daemon is started or stopped as following.

[STARTING]

```
# systemctl start nqs-btu.service
```

[STOPPING]

```
# systemctl stop nqs-btu.service
```

Boot-up daemon executes the following four shell scripts according to the situation. These shell scripts need to be prepared beforehand being tailored to the environment of your system.

1. Configuration script of boot-up environment (/opt/nec/nqsv/sbin/btu_prog/btup_configure.sh)
2. Boot-up script of redundancy target (/opt/nec/nqsv/sbin/btu_prog/btup_bootup.sh)
3. Stop script of boot-up environment (/opt/nec/nqsv/sbin/btu_prog/rlbk_configure.sh)
4. Stop script of redundancy target (/opt/nec/nqsv/sbin/btu_prog/rlbk_bootup.sh)

Details of these scripts are described later.

### 20.1.2. Redundancy Function Settings

(1) Preliminary preparations

When Redundancy Function is used, do the following configuration on the BSV host.

- Assign machine ID to host name corresponding to virtual IP address
- Configure shared disk so that things under /var/opt/nec/nqsv are shared
- Share things under /etc/opt/nec/nqsv where definition files are placed. However, they can exist on local disk if they are synchronized manually
- Auto start and stop by systemd for the substituted component need to be disabled.

On the primary host, Start Batch server by specifying its virtual IP address to -a option of nqs_bsvd command.

```
# /opt/nec/nqsv/sbin/nqs_bsvd -a <Virtual_IP_Address>
```

When virtual IP address is specified, Batch server works as below.

- When Batch server, Routing server and Staging server that are started by Batch server make a TCP connection to remote processes, virtual IP address is used as local address. From the viewpoint of remote processes, IP address of Batch server is always the virtual IP address by this. Therefore, IP address is always kept the same IP address between the failover nodes.
- Batch server host name returned by NQSV/API is the one corresponding to virtual IP address. Thus, host name displayed as Batch server name by client such as CUI command, Scheduler, etc. using NQSV/API is the host name corresponding to virtual IP address. Host name is also always kept the same IP address between the failover nodes.

On the stand-by host, edit the BSV_PARAM in /etc/opt/nec/nqsv/nqs_bsv.env file as follows to inherit the virtual IP address.

```
BSV_PARAM="-a ${VIRTUAL_IP_ADDRESS}"
```

(2) Boot-up Daemon Settings

Boot-up daemon wait the order to boot-up the components on the substitute host from node agent. After receive the order to boot-up, it execute the following script for boot-up the components which is created on the substitute host.

| Type of script | Default Path |
|---|---|
| Boot-up environment configuration script | /opt/nec/nqsv/sbin/btu_prog/btup_configure.sh |
| Redundancy target booting up script | /opt/nec/nqsv/sbin/btu_prog/btup_bootup.sh |
| Boot-up environment stopping script | /opt/nec/nqsv/sbin/btu_prog/rlbk_configure.sh |

| | |
|---|---|
| Redundancy target stopping script | /opt/nec/nqsv/sbin/btu_prog/rlbk_bootup.sh |

For Boot-up daemon, path name of 4 shell scripts and TCP port number can be changed by specifying start-up options.

```
# nqs_btud [-p <port>] [-c <1_path>] [-b <2_path>] [-C <3_path>] [-B <4_path>]
```

*<port>* is TCP port number used by Boot-up daemon. Port number can range between 0 and 65535. If it is not set or it is set to value out of range, default is 49600.

*<1_path> <2_path> <3_path> <4_path>* are path name of shell script, respectively. The maximum length of path name is 1023 characters. If path name exceeds the maximum length, default value of path name is used.

| Item | Content | Default Value |
|---|---|---|
| *<1_path>* | Boot-up environment configuration script | /opt/nec/nqsv/sbin/btu_prog/btup_configure.sh |
| *<2_path>* | Redundancy target booting up script | /opt/nec/nqsv/sbin/btu_prog/btup_bootup.sh |
| *<3_path>* | Boot-up environment stopping script | /opt/nec/nqsv/sbin/btu_prog/rlbk_configure.sh |
| *<4_path>* | Redundancy target stopping script | /opt/nec/nqsv/sbin/btu_prog/rlbk_bootup.sh |

**◆Boot-up environment configuration script**

This script configures environment for substitute host to add virtual IP address and/or to mount shared file system where database is deployed. Virtual IP address is passed as an argument to script. Configuration example is as follows.

```
#!/bin/sh
# sample /opt/nec/nqsv/sbin/btu_prog/btup_configure.sh
export PATH=/sbin:/bin:/usr/bin
# mount NFS for configuration files & database files
case $1 in
192.168.1.1)
    mount -t nfs nfs_server:/exports/etc/opt/nec/nqsv/host-a /etc/opt/nec/nqsv || exit $?
    mount -t nfs nfs_server:/exports/var/opt/nec/nqsv/host-a /var/opt/nec/nqsv || exit $?;;
192.168.1.2)
    mount -t nfs nfs_server:/exports/etc/opt/nec/nqsv/host-a /etc/opt/nec/nqsv || exit $?
```

```
	mount -t nfs nfs_server:/exports/var/opt/nec/nqsv/host-b /var/opt/nec/nqsv || exit $?;;
192.168.1.3)
	mount -t nfs nfs_server:/exports/etc/opt/nec/nqsv/host-c /etc/opt/nec/nqsv || exit $?
	mount -t nfs nfs_server:/exports/var/opt/nec/nqsv/host-c /var/opt/nec/nqsv || exit $?;;
esac
# add secondary ip address / subnetmask to eth0 temporarily
ip address add $1/24 dev eth0
```

◆**Redundancy target booting up script**

This script starts up redundancy target of batch server and so forth. Virtual IP address is passed by environment variable VIRTUAL_IP_ADDRESS. Description example is as follows.

```
#!/bin/sh
# sample /opt/nec/nqsv/sbin/btu_prog/btup_bootup.sh
export PATH=/bin:/usr/bin
if ps -ef | fgrep -q '/opt/nec/nqsv/sbin/nqs_bsvd'
then
	systemctl stop nqs-bsv.service
fi
systemctl start nqs-bsv.service || exit $?
if ps -ef | fgrep -q '/opt/nec/nqsv/sbin/nqs_jmd'
then
	systemctl stop nqs-jmd.service
fi

systemctl start nqs-jmd.service
```

◆**Boot-up environment stopping script**

This script stops substitute host. It deletes virtual IP address and unmount shared file system where database is deployed. Virtual IP address is passed as an argument to script. Description example is as follows.

```
#!/bin/sh
# sample /opt/nec/nqsv/sbin/btu_prog/rlbk_configure.sh
export PATH=/sbin:/bin:/usr/bin
# add secondary ip address / subnetmask
if ip address show dev eth0 | fgrep -q -w "$1"
then
	ip address del $1/24 dev eth0 || exit $?
fi
```

```
# unmount NFS for configuration files & database files
if mountpoint -q /var/opt/nec/nqsv
then
	umount /var/opt/nec/nqsv || exit $?
fi
if mountpoint -q /etc/opt/nec/nqsv
then
	umount /etc/opt/nec/nqsv
fi
```

◆**Redundancy target stopping script**

This script stops redundancy target of batch server. Virtual IP address is passed by environment variable VIRTUAL_IP_ADDRESS. Description example is as follows.

```
#!/bin/sh
# sample /opt/nec/nqsv/sbin/btu_prog/rlbk_bootup.sh
export PATH=/bin:/usr/bin
if ps -ef | fgrep -q '/opt/nec/nqsv/sbin/jmd'
then
	systemctl stop nqs-jmd.service || exit $?
fi
if ps -ef | fgrep -q '/opt/nec/nqsv/sbin/nqs_bsvd'
then
	systemctl stop nqs-bsv.service
fi
```

(3) Node Management Agent Settings

Redundancy function uses node management agent. (Please refer to 18.3. Failure Detection Function Settings

### 20.1.3. Failure Detection by Simplified Failure Detection Script for Redundancy Function

You can detect the failure of the host by using the simplified failure detection script without system management software like Zabbix. It is installed to following path if node management agent is installed.

Simplified failure detection script: /opt/nec/nqsv/sbin/check_alive.sh

This section describes about detecting failure by using this script.

Please set target host to be monitored, BMC of the host, BMC user name and BMC password

after the "# HostList" line by editing the simplified failure detection script, /opt/nec/nqsv/sbin/check_alive.sh on node management host.

```
#!/bin/bash
export PATH=/bin:/usr/bin:/opt/nec/nqsv/sbin/nqsv
tempfile=`mktemp`
sed '1,/^# HostList/d; /#/d' $0 > $tempfile
exec < $tempfile
while read host bmc user pw
do
	if ping -c 4 $host > /dev/null 2>&1
	then
		:
	else
		nqs_ntfr -s $host
		ipmitool -I lanplus -H $bmc -U $user -P $pw chassis power off
	fi
done
rm $tempfile
exit 0
# HostList
# Information of the hosts to check alive are written in following lines.
# Physical_IP_Address BMC_IP_Address BMC_User_Name BMC_Password
host-a	host-a-bmc	bmc-a-user	bmc-a-pw
host-b	host-b-bmc	bmc-b-user	bmc-b-pw
```

In addition, on node management host, cron needs to be configured so that this simplified failure detection script is periodically executed by cron. This enables simplified failure detection script to periodically check whether batch server host is alive and to notify node management agent of failure if it cannot connect with batch server host.

Node management agent which receives failure notification from simplified failure detection script switches the failed host to a substitute host.

(1) Failure Detection by OSS

Failure notification command, nqs_ntfr is used in order to detect failure by using OSS. Failure notification command notifies node management agent of failures and node management agent starts redundancy processing.

Failure notification command is installed in following path of node management agent host.

Failure notification command:　/opt/nec/nqsv/sbin/nqs_ntfr

Its format is as follows.

```
nqs_ntfr -s hostname | IP_address
```

nqs_ntfr command notifies node management agent of failures about execution host, where redundancy target such as batch server works, specified by hostname or IP_address.

(2)　Failure Record

When failures are notified, they are recorded in the log file of node management agent, /var/opt/nec/nqsv/node_agent_log.

> [Note]
> In case that log level is set to 0, all of failure notifications are not necessarily recorded. This is because failure notification itself is just normal process of node management agent, so it is not fallen into neither error nor warning as long as the failure notification is processed normally. Therefore, it is not recommended to set log level to 0.

Once failure is notified and failed host is normally switched to a substitute host, the following message is to be recorded. (Date and time is actually recorded at the beginning of a message) xxx.xxx.xxx.xxx is physical IP address of failed host and yyy.yyy.yyy.yyy is physical IP address of a substitute host. This message is not recorded at log level of 0.

```
[INFO   ] IP: xxx.xxx.xxx.xxx was switched. => IP: yyy.yyy.yyy.yyy
```

When there is no substitute host available even though failure occurs, the following message is to be recorded. xxx.xxx.xxx.xxx is physical IP address of failed host.

```
[ERROR ] no available waiting node.. (IP: xxx.xxx.xxx.xxx)
```

When power on of a substitute host switched to fails, the following message is to be recorded. xxx.xxx.xxx.xxx is physical IP address of failed host. At this time, if another substitute host is available, the substitute host is used.

```
[ERROR ] failed to power on. (IP: xxx.xxx.xxx.xxx)
```

When a substitute host switched to doesn't respond, the following message is to be recorded. xxx.xxx.xxx.xxx is physical IP address of failed host. At this time, if another substitute host is available, the substitute host is used.

```
[ERROR ] no response waiting node. (IP: xxx.xxx.xxx.xxx)
```

When boot-up of a substitute host switched to fails, the host is automatically stopped. If the

host is successfully stopped, the following message is to be recorded. xxx.xxx.xxx.xxx is physical IP address of failed host. At this time, if another substitute host is available, the substitute host is used.

```
[ERROR ] failed to boot up. (IP: xxx.xxx.xxx.xxx)
```

When boot-up process fails and then the stop also fails, the following message is to be recorded. xxx.xxx.xxx.xxx is physical IP address of failed host. In this case, even if another substitute host is available, the substitute host is not used.

```
[ERROR ] failed to boot up and roll back. (IP: xxx.xxx.xxx.xxx)
```

### 20.1.4. Failed Host Recovery

(1) Status Check

Node Management Agent records the latest status of redundancy to status record file. Path of status record file is following.

Status record file: /var/opt/nec/nqsv/nag_redundancy

At the timing of recording messages into a log file, status record file is updated. Latest status is always recorded in this file since it is overwritten each time. Regardless of any log level, the file is updated.

> [Note]
> This file is created when any failure is notified for the first time. So, the file does not exist if no failure has occurred.
> Do not edit this file manually. It is used for internal processing of node management agent too.
> If there is any conflict between nag_backup.conf and nag_redundancy file, nag_redundancy file is renamed to "nag_redundancy.rej" and new nag_redundacy file is created.

Path name of status record file can be changed by specifying the start-up option of node management agent.

```
nqs_nagd -R <path>
```

Status record file is recorded in the following format. Each record has a set of three information such as (1) a list of failed hosts, (2) header, and (3) status of substitute host(s).

A set of the information corresponds to a pair of managed host and substitute host in the setting file for managed host and substitute host. Header is just explanation about status of substitute host, so it has no significance in itself. A set of the three information is repeatedly

recorded as many as those information exist. A line for a list of failed hosts invariably exists in each set. Status of a substitute host is recorded in a line, so lines of status of substitute hosts exist as many as the number of substitute hosts.

```
Date and time of record
A list of failed host(s)
Header
Status of substitute host
A list of failed host(s)
Header
Status of substitute host
    :
```

A list of failed host(s) is recorded in the following format. When there is no failed host, "none" is recorded. When there exist multiple failed hosts, each host is recorded with a blank delimiter.

```
defected hosts= <Physical IP address> ... | (none)
```

Status of substitute host is recorded in the following format. The part in square brackets, [] is to be recorded only when status is running.

```
<Physical IP address> <Status description> [<Physical IP address of active host switched to>]
```

There are 5 types of status and each status has the following meaning.

| Status | Meaning |
|---|---|
| waiting | Switching hosts is not done yet and it is in a waiting state. |
| switching | It is in the process of switch. |
| switched | Switching hosts is done and it is in a switched state. |
| recovering | It is in the process of recovery. |
| failed | Either switching hosts fails or stopping host fails. The substitute host cannot be used. |

Output example of status record file is as follows. In this example, some failure occurred to the host,192.168.1.100 and it was switched to the host, 192.168.1.200. In addition, another failure occurred to the host, 192.168.1.101 but it is not switched to any host. Some kind of error occurred to the substitute host, 192.168.1.201, so it cannot be used as a substitute host.

```
11/11 11:11:11
defected hosts= 192.168.1.100 192.168.1.101
# ip waiting host     state          ip switched host
```

```
192.168.1.200      switched      192.168.1.100
192.168.1.201      failed
192.168.1.202      waiting
defected hosts= (none)
# ip waiting host     state          ip switched host
192.168.1.203      waiting
```

(2) Failed Host Recovery

Procedure to return the failed host to operation after the host is repaired is explained.

The procedure is divided into 2 stages.

1. Stop substitute host with nqs_ntfr -r command.
2. Notify the completion of recovering to node management agent with nqs_ntfr -R command.

It is illustrated as an example of recovering the host, 192.168.1.100 which is exemplified in status record file in the section, "(1) Status Check".

```
[Note]
There is period that former operation host and substitute host don't work.
```

First, please stop the substitute host by using failure notification command. This operation needs to be done on node management host. The specified host name or IP address here is not the one of substitute host but the former host returned to operation.

```
# nqs_ntfr -r <Host name> | <IP address>
```

In this example, the following command is executed.

```
# nqs_ntfr -r 192.168.1.100
```

When the host is successfully stopped, the following message is recorded. xxx.xxx.xxx.xxx below is physical IP address of substitute host. However, it is not recorded at log level of 0.

```
[INFO   ] succeeded to stop. (IP: xxx.xxx.xxx.xxx)
```

In addition, status record file is updated at the same time. Status of corresponding host changes to waiting state.

```
11/17 17:17:17
defected hosts= 192.168.1.100 192.168.1.101
# ip waiting host     state          ip switched host
192.168.1.200      waiting
```

```
192.168.1.201      failed
192.168.1.202      waiting
defected hosts= (none)
# ip waiting host     state          ip switched host
192.168.1.203      waiting
```

If any trouble happens here, please refer following "Trouble shooting when stopping substitute host".

After confirming substitute host is stopped, start the host to be returned to operation. This operation needs to be done by administrator.
After the host returned to operation is successfully started, notify recovery completion by using failure notification command. Host name specified here is the one of the former host returned to operation.

```
# nqs_ntfr -R <Host name> | <IP address>
```

In this example, the following command is executed.

```
# nqs_ntfr -R 192.168.1.100
```

The following message is recorded in log file. xxx.xxx.xxx.xxx below is physical IP address of the host returned to operation. However, it is not recorded at log level of 0.

```
[INFO   ] IP: xxx.xxx.xxx.xxx was recovered.
```

By checking status record file, it is found that the host, 192.168.1.100 is removed from the list of failed host. It means failed host recovery is now complete.

```
11/17 18:00:00:00
defected hosts= 192.168.1.101
# ip waiting host     state          ip switched host
192.168.1.200      waiting
192.168.1.201      failed
192.168.1.202      waiting
defected hosts= (none)
# ip waiting host     state          ip switched host
192.168.1.203      waiting
```

**Trouble shooting when stopping substitute host**

1) Status of corresponding host is in recovering state

After execute nqs_ntfr -r command, if status of corresponding hosts is in recovering state

as follows, recovery is still processing. Please wait status to change into waiting or failed described later.

```
11/17 17:17:17
defected hosts= 192.168.1.100 192.168.1.101
# ip waiting host     state          ip switched host
192.168.1.200       recovering
192.168.1.201       failed
192.168.1.202       waiting
defected hosts= (none)
# ip waiting host     state          ip switched host
192.168.1.203       waiting
```

2) Status record file is not updated at all

If status record file is not updated at all, it is possible that host name specified by nqs_ntfr -r is wrong or the host is under the control of other node management agent. Please check the host name specified and/or path name of status record file.

3) Stop of substitute host failed

When the stop of substitute host by executing nqs_ntfr -r fails for some reason, the following message is recorded in log file. xxx.xxx.xxx.xxx below is physical IP address of the substitute host failed to stop.

```
[ERROR ] failed to stop. (IP: xxx.xxx.xxx.xxx)
```

In addition, status record file is updated as follows.

```
11/17 17:17:17
defected hosts= 192.168.1.100 192.168.1.101
# ip waiting host     state          ip switched host
192.168.1.200       failed         192.168.1.100
192.168.1.201       failed
192.168.1.202       waiting
defected hosts= (none)
# ip waiting host     state          ip switched host
192.168.1.203       waiting
```

In this case, shutdown substitute host 192.168.1.200 manually, and boot primary host 192.168.1.100.

After booting primary host, notify recovery completion by using failure notification command with -R option. Host name specified here is the primary host returned to

operation.

```
# nqs_ntfr -R 192.168.1.100
```

The status record file is updated as follows. The recovery is finished.

```
11/17 18:00:00
defected hosts= 192.168.1.101
# ip waiting host     state          ip switched host
192.168.1.200       failed
192.168.1.201       failed
192.168.1.202       waiting
defected hosts= (none)
# ip waiting host     state          ip switched host
192.168.1.203       waiting
```

To recover substitute host 192.168.1.200, please refer to "(4).Recovery of Error Occurred Substitute Host"

(3) Recovery of Failed Host not substituted

Recovery procedure is illustrated below as an example of recovering the host, 192.168.1.101 which is exemplified in status record file in the section, "(1) Status Check".

[Note]
It is recommended to shut down all substitute hosts with "failed" status in status record file.

After the host returned to operation is successfully started, notify recovery completion by using failure notification command. Host name specified here is the one of the former host returned to operation. In this example, the following command is executed.

```
# nqs_ntfr -R 192.168.1.101
```

By checking status record file, it is found that the host, 192.168.1.101 is removed from the list of failed host. It means failed host recovery is now complete.

```
11/17 17:17:17
defected hosts= 192.168.1.100
# ip waiting host     state          ip switched host
192.168.1.200       switched       192.168.1.100
192.168.1.201       failed
```

```
192.168.1.202      waiting
defected hosts= (none)
# ip waiting host     state          ip switched host
192.168.1.203      waiting
```

(4) Recovery of Error Occurred Substitute Host

Once the cause of error of substitute host with failed status is found out and the error is resolved, the host is ready to be used again as substitute host. Recovery procedure is illustrated below as an example of recovering the host, 192.168.1.201 which is exemplified in status record file in the section, "(1) Status Check"..

After the error is successfully resolved, notify recovery completion by using failure notification command. Host name specified here is the one of substitute host returned to operation. In this example, the following command is executed

```
# nqs_ntfr -R 192.168.1.201
```

By checking status record file, it is found that status of the host, 192.168.1.201 is changed into warning state. It means error occurred host recovery is now complete.

```
11/17 17:17:17
defected hosts= 192.168.1.100
# ip waiting host     state          ip switched host
192.168.1.200      switched      192.168.1.100
192.168.1.201      waiting
192.168.1.202      waiting
defected hosts= (none)
# ip waiting host     state          ip switched host
192.168.1.203      waiting
```

## 20.2. Redundancy Function using EXPRESSCLUSTER

A batch server, accounting server, accounting monitor and JobManipulator corresponds to duplication by EXPRESSCLUSTER, and it is possible to continue NQSV system service.
The following EXPRESSCLUSTER versions have been confirmed to work with NQSV.

| EXPRESSCLUSTER version | Verified OS |
|---|---|
| EXPRESSCLUSTER X 4.3 | Red Hat Enterprise Linux 7.9<br>CentOS 7.9 |
| EXPRESSCLUSTER X 5.0 | Red Hat Enterprise Linux 7.9, 8.6, 8.8<br>CentOS 7.9<br>Rocky Linux 8.6, 8.8 |

### 20.2.1. Notes

When using NQSV on EXPRESSCLUSTER, please be careful about the following point.

- Specify a floating IP address by -a option of batch server and JobManipulator.

```
# /opt/nec/nqsv/sbin/nqs_bsvd -a <Floating_IP_Address>
# /opt/nec/nqsv/sbin/nqs_jmd -a <Floating_IP_Address>
```

- Specify the virtual host name of the Accounting server associated with floating IP address, for the Accounting server setting if duplicate the Accounting server.

- If duplicate the Accounting server, duplicate the Accounting monitor too.

- Allocate a machine ID to the host name that corresponds to a floating IP address.

- /var/opt/nec/nqsv where a batch server database exists should be shared using by shared volume.

- Systemd service for a batch server and a scheduler controlled by EXPRESSCLUSTER need to be invalidated for booting or Stopping time. Otherwise, shutdown of the stand-by system can make components on the active system stop.

- /var/opt/nec/nqsv where the data base exist, and /etc/opt/nec/nqsv where setting files exist should be shared. In particular, move these 2 directories on the shared file system, and create the symbolic link to original /var/opt/nec/nqsv and /etc/opt/nec/nqsv.

[Notes on EXPRESSCLUSTER]
If you are using nas for disk resources in EXPRESSCLUSTER X 4.3 or earlier, you cannot use nas for disk resources in EXPRESSCLUSTER X 5.0.

### 20.2.2. Configurations

This section describes the duplication of NQSV. Parameters that are not listed all have their default settings. The following is setting on "Builder" included in EXPRESSCLUSTER modules. Please refer the "Installation and Configuration Guide" and "Reference Guide" of EXPRESSCLUSTER X for installation and usage of EXPRESSCLUSTER Builder.

(1) Create failover group

First, create failover group for NQSV on EXPRESSCLUSTER as following.

| name | NQSVSOFT(Optional) |
|---|---|
| Type | Failover |
| Startup server | [Failover is possible at all servers] conditions of default |
| Attribute | Condition of default |

Next, add the following resource to the NQSV group.

| name | DK_NQSV(Optional) |
|---|---|
| type | disk resource |
| Dependency | Follow the default dependence |
| File system | ext4 |
| Device name | /dev/sdb1(*) |
| Mountpoint | /var/opt/nec/nqsv |

*Specify the appropriate device name according to the environment.

Next, add the following resource to the NQSV group.

| name | FIP_NQSV (Optional) |
|---|---|
| type | floating ip resource |

| name | FIP_NQSV (Optional) |
|---|---|
| Dependency | Follow the default dependence |
| IP address | 10.34.154.138 (*) |

*Specify the appropriate IP address according to the environment.

(2) Adding exec resources

Add exec resources used in NQSV to NQSVSOFT group as follows. On this section, describes about duplication of a batch server, but you can change the parameter for the Accounting server, Accounting monitor and JobManipulator.

(exec resources for starting and stopping NQSV and JobManipulator)

| setting items | | | | setting parameters |
|---|---|---|---|---|
| Info | | | Name | EXE_BSV |
| | | | Type | execute resource |
| Dependency | | | | Follow the default dependence |
| Details | Script list | | Start script | start.sh (*1) |
| | | | Stop script | stop.sh (*1) |
| | Tuning | Parameter | Start script | ■Synchronous □Asynchronous |
| | | | Stop script | ■Synchronous □Asynchronous |

*1 : Script contents are listed in (4) (1)(2)

(exec resource for NQSV and JobManipulator process monitoring)

| setting items | | | setting parameters |
|---|---|---|---|
| Info | | Name | EXE_BSVMON |
| | | Type | execute resource |
| Dependency | Dependent Resources | | DK_NQSV<br>FIP_NQSV<br>EXE_BSV |
| Details | Script list | Start script | start.sh (*1) |
| | | Stop script | stop.sh (*2) |

| setting items | | | | setting parameters |
|---|---|---|---|---|
| | Tuning | Parameter | Start script | □Synchronous ■Asynchronous |
| | | | Stop script | ■Synchronous □Asynchronous |

*1 : Script contents are listed in (4) (3)

*2 : Do not replace stop.sh. Please use a script already set as default.

(3) Monitor resource setup

Next, add the PID monitor resources for NQSV. On this section, describes about a batch server, but you can change the parameters for the Accounting server, Accounting monitor and JobManipulator.

(PID monitor resource or BSV process)

| setting items | | setting parameters |
|---|---|---|
| Info | Name | pidw-BSVMON |
| | Type | pid monitor |
| Monitor(Common) | Target Resource | EXE_BSVMON |
| Recovery action | Recovery Target | NQSVSOFT |

Add the following monitor resources.

(Disk monitor resource)

| setting item | | setting parameter |
|---|---|---|
| Info | Name | diskw-NQSVSOFT |
| | Type | disk monitor |
| Monitor | Method | READ(O_DIRECT) |
| | Target Device Name | /dev/sdb1(*) |
| Recovery action | Recovery Target | NQSVSOFT |

*Specify the appropriate Target Device Name according to the environment.

(NIC Link UP/Down monitor resource)

| setting item | | setting parameters |
|---|---|---|
| Info | Name | miiw-NQSVSOFT |
| | Type | NIC Link Up/Down monitor |
| Monitor | Target | eth0 (*) |
| Recovery action | Recovery Target | NQSVSOFT |

*Specify the appropriate Target according to the environment.

(4) Scripts of exec resource

Following scripts are used for setting. These scripts process for batch server, but you can change the processes for the Accounting server, Accounting monitor and JobManipulator.

**i) start.sh of EXE_BSV**

```
#! /bin/sh
#*************************************
#*              start.sh              *
#*************************************

if [ "$CLP_EVENT" = "START" ]
then
    if [ "$CLP_DISK" = "SUCCESS" ]
    then
            echo "NORMAL1"
            ###### for BSV software ######
            systemctl start nqs-bsv.service
            ret=$?
            if [ $ret -ne 0 ]; then
                    exit $ret;
            fi;
            ################################
            if [ "$CLP_SERVER" = "HOME" ]
            then
                    echo "NORMAL2"
            else
                    echo "ON_OTHER1"
            fi
    else
            echo "ERROR_DISK from START"
    fi
elif [ "$CLP_EVENT" = "FAILOVER" ]
then
    if [ "$CLP_DISK" = "SUCCESS" ]
    then
            echo "FAILOVER1"
            ###### for BSV software ######
```

```
            systemctl start nqs-bsv.service
            ret=$?
            if [ $ret -ne 0 ]; then
                    exit $ret;
            fi;
            ##############################
            if [ "$CLP_SERVER" = "HOME" ]
            then
                    echo "FAILOVER2"
            else
                    echo "ON_OTHER2"
            fi
    else
            echo "ERROR_DISK from FAILOVER"
    fi
else
    echo "NO_CLP"
fi
echo "EXIT"
exit 0
```

**ii)　stop.sh of EXE_BSV**

```
#! /bin/sh
#*************************************
#*              stop.sh              *
#*************************************

if [ "$CLP_EVENT" = "START" ]
then
    if [ "$CLP_DISK" = "SUCCESS" ]
    then
            echo "NORMAL1"
            ##### For BSV Software #####
            systemctl stop nqs-bsv.service
            ret=$?
            if [ $ret -ne 0 ]; then
                    exit $ret;
            fi;
            ############################
            if [ "$CLP_SERVER" = "HOME" ]
            then
                    echo "NORMAL2"
            else
                    echo "ON_OTHER1"
            fi
    else
            echo "ERROR_DISK from START"
    fi
elif [ "$CLP_EVENT" = "FAILOVER" ]
```

```
then
    if [ "$CLP_DISK" = "SUCCESS" ]
    then
            echo "FAILOVER1"
            ##### For BSV Software #####
            systemctl stop nqs-bsv.service
            ret=$?
            if [ $ret -ne 0 ]; then
                    exit $ret;
            fi;
            ############################
            if [ "$CLP_SERVER" = "HOME" ]
            then
                    echo "FAILOVER2"
            else
                    echo "ON_OTHER2"
            fi
    else
            echo "ERROR_DISK from FAILOVER"
    fi
else
    echo "NO_CLP"
fi
echo "EXIT"
exit 0
```

**iii） start.sh of EXE_BSVMON**

```
#! /bin/sh
#*************************************
#*              start.sh              *
#*************************************

if [ "$CLP_EVENT" = "START" ]
then
    if [ "$CLP_DISK" = "SUCCESS" ]
    then
            echo "NORMAL1"
            ##### start BSV MONITOR #####
            /var/opt/nec/nqsv/bsvmon.sh
            ret=$?
            if [ $ret -ne 0 ]; then
                    exit $ret
            fi;
            ##################################
            if [ "$CLP_SERVER" = "HOME" ]
            then
                    echo "NORMAL2"
            else
                    echo "ON_OTHER1"
            fi
```

```
    else
            echo "ERROR_DISK from START"
    fi
elif [ "$CLP_EVENT" = "FAILOVER" ]
then
    if [ "$CLP_DISK" = "SUCCESS" ]
    then
            echo "FAILOVER1"
            ##### start BSV MONITOR #####
            /var/opt/nec/nqsv/bsvmon.sh
            ret=$?
            if [ $ret -ne 0 ]; then
                    exit $ret
            fi;
            ##################################
            if [ "$CLP_SERVER" = "HOME" ]
            then
                    echo "FAILOVER2"
            else
                    echo "ON_OTHER2"
            fi
    else
            echo "ERROR_DISK from FAILOVER"
    fi
else
    echo "NO_CLP"
fi
echo "EXIT"
exit 0
```

**iv) Contents of /var/opt/nec/nqsv/bsvmon.sh**

```
#!/bin/sh
#######################################################################
#
# Name: /var/opt/nec/nqsv/bsvmon.sh
#
#######################################################################

BSV_PROC[0]="/opt/nec/nqsv/sbin/nqs_bsvd"

# The interval (in seconds) between checking
# if processes of BSV are up and running.
INTERVAL=60

while :
do
        for PROC in ${BSV_PROC[@]}
        do
                pid=`ps -ef | grep "$PROC" | grep -v grep`
                if [ -z "$pid" ]; then
```

```
                        echo "NOTICE: $PROC is terminated"
                        exit 1
                fi
        done

        sleep $INTERVAL
done
```

### 20.2.3. How to start and stop NQSV services

This section describes how to start and stop the NQSV services in EXPRESSCLUSTER. After configuration, please start and stop the NQSV services by operating the resource of EXPRESSCLUSTER, not by using the systemctl command. Not doing so may result in failover.

The documentation below describes how to start and stop the BSV service as an example: start/stop the exec resource and the process monitor resource as follows.

- Start of service
  (1) Start the exec resource "EXE_BSV".
  (2) Start the resource "EXE_BSVMON" that monitors the process of BSV service.
  The startup of this resource will also automatically run "pidw-BSVMON".

- Stop of service
  (1) Stop the resource "EXE_BSVMON" that monitors the process of BSV service.
  Stopping this resource will also automatically terminate "pidw-BSVMON".
  (2) Stop the exec resource "EXE_BSV".
  The stopping of this resource terminates BSV service.

# 21. Using OSS for Batch Job Collaboration

NQSV can work with OSS (e.g., Dask-Jobqueue, ClusterMQ, etc.) to submit and execute jobs from R or Python directly into the batch job system.

## 21.1. Environment Settings

To link to job submission from OSS, set the NQSV linkage command to be used as a symbolic link instead of the SLURM command used in OSS. This enables NQSV commands to be invoked to submit, display, and delete jobs when jobs are submitted and executed from OSS. Create a symbolic link as shown below.

| Command name | Symbolic link destination | Description |
|---|---|---|
| sbatch | /opt/nec/nqsv/bin/sbatch_cnv.sh | This is a batch job submission command.<br>This command converts the I/F of the sbatch command of SLURM to the I/F of the qsub(1) command of NQSV to submit a batch job. |
| squeue | /opt/nec/nqsv/bin/squeue_cnv.sh | Batch job display command.<br>It converts the I/F of the squeue command of SLURM to the I/F of the qstat(1) command of NQSV to display a batch job. |
| scancel | /opt/nec/nqsv/bin/scancel_cnv.sh | This is a batch job deletion command.<br>It converts the I/F of the scancel command of SLURM to the I/F of the qstat(1) command of NQSV to delete a batch job. |

When using this function, please create the symbolic link above manually. The location of the symbolic link file should be the path of the batch job command to be used from the OSS.
For details on how to use each OSS, please refer to the OSS website.

## Appendix.A Use Case

### A-1 Differences in the functions of running scripts

The table below describes the differences between the four functions.

- User Exit
- Hook Script Function
- User Pre-Post Script Function
- Resource Monitoring Script of Custom Resource function

| | Authority | When to run | Where to run | Purpose of use | Remarks |
|---|---|---|---|---|---|
| User EXIT | Root | PRE-RUNNING<br>POST-RUNNING<br>HOLDING<br>RESTARTING | JobServer host | Clearing the work area | wait for the script completion<br><br>handling the script execution result |
| Hook Script | Root | QUEUED<br>WAITING<br>HELD<br>HOLDING<br>SUSPENDING<br>SUSPENDED<br>STAGING<br>ARRIVING<br>PRE-RUNNING<br>RUNNING<br>POST-RUNNING<br>EXITING<br>EXITED<br>RESUMING<br>etc | BatchServer host | Reject a job specified with wrong parameters | cannot wait for the script completion<br><br>cannot handling the script execution result |

| | | | | | |
|---|---|---|---|---|---|
| User Pre-Post Script | User | PRE-RUNNING POST-RUNNING | JobServer host | Health check | |
| Resource Monitoring Script | root | RUNNING | JobServer host | Counting resource usage of Custom Resource | |

**A-2 Example: Limits on standard output and error output files (User-Exit)**

This is an example of a script that prevent the files transfer when the standard output and error output file size is too large.

This script should be run in POST-RUNNING.

After job running, if files size exceeds 30MB, save the files to /tmp/backup_dir and following message outputs.

"The size of stdout(stderr) exceed 31457280 byte(s).“

```
#!/bin/sh
FILESIZELIMIT=31457280
BACKUP_DIR_STDOUT=/tmp/backup_dir
BACKUP_DIR_STDERR=/tmp/backup_dir

case "$UEX_LOCATION" in
POSTRUNNING)
    case "$UEX_PROCTYPE" in
    EXECUTION)
        JOBID=`echo $UEX_STGDIR | sed 's|.*/\(.*\)/.*|\1|'`
        STDOUT=/var/opt/nec/nqsv/jsv/jobfile/$JOBID/stdout
        STDERR=/var/opt/nec/nqsv/jsv/jobfile/$JOBID/stderr

        [ -z "$PBS_JOBNAME" ] && echo 1>&2 "PBS_JOBNAME is not set" && exit 1
        [ -z "$UEX_STGDIR" ] && echo 1>&2 "UEX_STGDIR is not set" && exit 1

        if [ `wc -c "$STDOUT" | awk '{print $1}'` -gt $FILESIZELIMIT ]; then
            if [ ! -d "$BACKUP_DIR_STDOUT" ]; then
                mkdir -p "$BACKUP_DIR_STDOUT" || exit 1
            fi
            mv $STDOUT $BACKUP_DIR_STDOUT/$PBS_JOBNAME.o$JOBID || exit 1
            echo "The size of stdout exceeded $FILESIZELIMIT byte(s)." > $STDOUT
        fi

        if [ `wc -c "$STDERR" | awk '{print $1}'` -gt $FILESIZELIMIT ]; then
            if [ ! -d "$BACKUP_DIR_STDERR" ]; then
                mkdir -p "$BACKUP_DIR_STDERR" || exit 1
            fi
            mv $STDERR $BACKUP_DIR_STDERR/$PBS_JOBNAME.e$JOBID || exit 1
```

```
                echo "The size of stderr exceeded $FILESIZELIMIT byte(s)." > $STDERR
            fi
            ;;
        esac
        ;;
    esac
```

### A-3 Example: Deleting the remaining processes (User-Exit)

This is an example of a script that delete processes that remained after job run in an interactive node.

This example is only available if the socket scheduling function is enabled.

This script should be run in POST-RUNNING.

```
#!/bin/bash

JID=`basename $UEX_JOBDB`

while read line
do
    kill -9 $line
done < /sys/fs/cgroup/cpuset/$JID/tasks
```

### In the case of the queue set RSG number

The memory cgroup path referenced by the user exit script is different. <cpuset_name> contains the CPUSET name corresponding to the RSG number set for the queue.

```
#!/bin/bash

JID=`basename $UEX_JOBDB`

while read line
do
    kill -9 $line
done < /sys/fs/cgroup/cpuset/<cpuset_name>/$JID/tasks
```

You can find CPUSET name by the following procedure.

First, you find RSG number of the queue for which you want to set UserExit. It is the number that is displayed in Resource Sharing Group of Kernel Parameter. It is 1 in the following example. If it is 0, RSG number is not set.

```
$ qstat -Qf intque
Interactive Queue: intque@bsvhost01
    Run State    = Active
    Submit State = Enable
      :
  Kernel Parameter:
    Resource Sharing Group     = 1
```

```
        Nice Value                  = 0
```

Next, you see /etc/opt/nec/nqsv/cpuset.conf in the execution host. The need item is the CPUSET name corresponding to the RSG number. It is "forjob" in the following example.

```
    $ cat /etc/opt/nec/nqsv/cpuset.conf
    CPUSET  0-31  0-1  0
    forjob  8-31  0-1  1
```

In this case, the UserExit script is following.

```
    #!/bin/bash

    JID=`basename $UEX_JOBDB`

    while read line
    do
        kill -9 $line
    done < /sys/fs/cgroup/cpuset/forjob/$JID/tasks
```

For more information on the CPUSET function, please refer to "18.2 CPUSET function" in this document.

### A-4 Example: Parameter check and job delete (hook script)

This is an example of a script that reject jobs which specify wrong parameters.

This script should be run in QUEUED or PRE-RUNNING.
If “-l cpunum_job=40” is specified but “-l gpunum_job=8” is not specified in the job, the job is deleted by qdel.

```
    #!/bin/bash

    CPUNUM=`qstat -f ${HOOK_RID} -Pm | grep "(Per-Job) CPU Number" | awk '{print
$6}'`
    GPUNUM=`qstat -f ${HOOK_RID} -Pm | grep "(Per-Job) GPU Number" | awk '{print
$6}'`

    if (CPUNUM == 40) ; then
       if (GPUNUM != 8) ; then
          qdel -Pm ${HOOK_RID}
       fi
    fi
```

### A-5 Example: Number of GPU (Resource Monitoring Script)

This is an example of a script that counts number of GPU which a job used.

This script should be run periodically in RUNNING.

If you use the script, create a custom resource that Consumer is job and Check mode is moment, put it to /opt/nec/nqsv/sbin/custom_prog/<custom resource name>.

```
#!/bin/bash

NVIDIA_SMI=/usr/bin/nvidia-smi
NVIDIA_SMI_OPT="pmon -c 1"
SAVED_FL=${CR_TMPDIR}/gpuacct

SUM=0
declare -A GPU_HASH

if [ -e ${SAVED_FL} ]; then
    while read line
    do
        GPU=`echo $line | awk '{print $1}'`
        GPU_HASH["$GPU"]=1
        SUM=$((${SUM} + 1))
    done < $SAVED_FL
fi

SMIOUT=`${NVIDIA_SMI} ${NVIDIA_SMI_OPT}`
if [ $? -ne 0 ];then
    exit 1
fi

while read line
do
    GPU=`echo $line | awk '{print $1}'`
    PID=`echo $line | awk '{print $2}'`
    if expr "${PID}" : "[0-9]*$" >&/dev/null; then
        SID=`ps --no-header -o sid -p ${PID} | awk '{print $1}'`
        if [ "$SID" = "$CR_EJID" ];then
            if [ -z ${GPU_HASH["$GPU"]} ];then
                GPU_HASH["$GPU"]=1
                SUM=$((${SUM} + 1))
            fi
        fi
    fi
done <<EOF
$SMIOUT
EOF

echo $SUM

if [ -e ${SAVED_FL} ]; then
    rm ${SAVED_FL}
fi

for g in "${!GPU_HASH[@]}"
do
    echo $g >> $SAVED_FL
done

exit 0
```

### A-6 Example: OSS Collaboration (Dask-Jobqueue)

This is an example of submitting and running a batch job using Dask-jobqueue's SLURMCluster()/cluster.scale() API in Python3 from the command line. For more details on each API and job submission/execution, please refer to the official OSS website.

```
$ python3
>>>
>>> from dask_jobqueue import SLURMCluster
>>> cluster = SLURMCluster(
queue='regular',
project="myproj",
cores=1,
memory="1 GB"
)
>>> cluster.scale(jobs=1)
>>>
```

## Appendix.B Update history

15th edition

- Correct an error in 12.5. HCA failure check

16th edition

- Added 2.3.20 Adjusting the Stage-Out Processing Buffer of the Request Result File
- Added 2.3.19 config parameters for memory cgroup
- Added 12.8 configuration for Multi-instance GPU(MIG)
- Added version EXPRESSCLUSTER X in 20.2

17th edition

- Added Stdout size limit and Stderr size limit in 4.1 and 4.5
- Added Extended submit limit for the number of VE nodes to section 12.3
- Added note on Dockerfile creation in 16.1 (4)
- Added 20.2.3 How to start and stop NQSV services
- Correct an error in A-5 Example

18th edition

- List the version of EXPRESSCLUSTER X confirmed to work in 20.2
- Added an example of user exit script when the queue is set RSG number.

19th edition

- Added 2.3.21 Change the behavior of submit request number limitayion exceeding during request routing.
- Added 5.1.7 Automatic rerun and billing exclusion function for HW failures.

# Index

## R



## S



## T



## U

## V

**NEC Network Queuing System V (NQSV) User's Guide**

**[Management]**

March 2024　19th edition

**NEC Corporation**