NQE Open Scheduling in a Cluster Environment

Janet Clegg and *Daryl Coulthart*, Cray Research, Inc.

ABSTRACT: *NQE 3.0 provides a significant improvement to workload scheduling for NQE customers. This new architecture allows greater flexibility in workload management and forms the basis for cluster-wide scheduling, parallel execution, and cluster-wide rerun of jobs. The model is centered around a network-based, SQL database and central scheduler. This paper will explain the new approach to NQE scheduling and show how customers can replace the NQE scheduler with a scheduler customized to their own needs.*

1 NQE Overview

The Network Queueing Environment (NQE) is a workload management product that schedules, monitors, and controls the execution of jobs in complex heterogeneous environments. NQE provides a consistent user interface across multiple heterogeneous UNIX systems. Users can submit their jobs to the NQE cluster, specify the resources needed for the job, and not worry about where the jobs will run. Based on the resources needed and the resources available in the NQE cluster, NQE will select an appropriate server, route the job request to that server, schedule and initiate the job request, and return STDOUT and STDERR files to the user.

2 NQE Workload Management

The NQE clients submit UNIX user shell scripts from UNIX systems to the NQE servers. The commands and displays are identical across all systems, so that users see a uniform environment and interface regardless of their workstation type. Motif based X-Window client programs display the status of jobs and the system load on each NQE server.

NQE contains a job scheduling program to route and schedule jobs and interactive commands to appropriate servers in the network cluster. Scheduling and selection is based on resources requested by the user and policies formulated by the site. These are policies and schedules implemented in the Network Load Balancer (NLB) and the NQE scheduler, which provide status and control of work scheduling, either balancing the network load and routing jobs to the best available server, or honoring requests for specific servers and resources.

Job requests may be submitted to the NQE scheduler which in turn will schedule the job for execution according to the NQE scheduler policy. Job requests can still be submitted directly to NQS servers or routed using NLB policies.

NQE's batch component, the Network Queueing system (NQS) provides fault tolerance and a rich set of administrative controls for high-end UNIX servers.

Job dependency gives users the ability to specify criteria which must be met before a job request will initiate. The most common application is to run a set of jobs in sequence.

NQE provides cluster-wide status and control of all jobs though the NQE GUI. All jobs owned by a user are visible regardless of what system they are on.

The File Transfer Agent (FTA) provides unattended file transfer across a network using the `ftp` protocol.

NQE uses FTA or `rcp` to return job output. Output from jobs is returned to the location requested by the user or to the current working directory from which the job request was made.

In addition, both an interactive utility (`ftua`) and a command line interface (`rft`) are available to initiate file transfers on NQE servers. The command line interface is particularly attractive when needed in batch job requests. Network Peer-to-Peer Authorization (NPPA) solves the problem of needing to supply passwords in job scripts or in .netrc files.

The user interface provides the ability to submit, display, status, signal, and batch jobs, using a command line interface or a GUI. The NQE GUI provides a consolidated user interface for NQE user functions.

3 NQE 3.0 Features

The NQE 3.0 release has the following new features:

- Integrated GUI for building and submitting job scripts, monitoring job status, controlling jobs, and monitoring NQE system status

- Display of FTA file transfer information in the GUI status displays
- GUI interface to cron, for launching predefined jobs at specific dates and times
- Ability to modify the scheduling of NQE job requests with Tcl (Tool Command Language)
- DCE/DFS integration, to provide NQE users with transparent access to their DFS files, while running NQE job scripts
- Cluster rerun, to automatically rerun jobs on another server if the execution server goes down
- Extensible collector to gather and display additional system information easily
- Interactive Load Balancing, to execute interactive UNIX commands on servers based on scheduling policies defined in the NLB
- Byte level recovery of FTA file transfers, so that file transfers are restarted from where they left off, in the event of network or system failures
- Support of new versions of IRIX, Solaris, and AIX UNIX operating systems

3.1 NQE Client/Server Model

NQE 3.0 provides a significant improvement to workload scheduling for NQE customers. This new architecture allows greater flexibility in workload management and forms the basis for parallel execution, cluster-wide scheduling, and cluster-wide rerun of jobs. The model is centered around a client/server SQL database and a central scheduler. This new NQE scheduler, available in 3.0, is written in the popular Tcl scripting language. This new feature allows sites to create a customized scheduler or to rework NQE's existing scheduler. For example, sites can elect to schedule based on NQE 2.0 style NLB policies, in which case the 3.0 Tcl scheduler calls the NLB policy module. But a site may instead choose to look at all jobs in the cluster and pick out a specific one to run first. With NQE 3.0 scheduling, sites now have the ability to enhance scheduling according to their individual needs.

3.2 DCE/DFS Integration

NQE provides the facility to perform DCE authentication at job initiation if a password is supplied with the `cqsub` command or the `nqe` GUI. If password validation is not in effect at the NQE server accepting the job, the password will only be used for DCE authentication and the usual NQS validation (such as file validation) will also take place.

If DCE authentication is successful, NQE supports access of DFS files within the user's DCE cell. This includes support of job scripts, job I/O, return of job output, and $HOME directories in DFS file space.

New DCE authentication will be obtained before restarting a checkpointed job on UNICOS.

DCE authentication is not required for status, signal, and delete operations.

3.3 NQE GUI

The `nqe` command invokes a new graphical user interface (GUI) that lets users do the following:

- Use the Submit window to open and edit a job script; to save changes made to a job script; to submit a request to NQE; to view, segment, delete, or reset the NQE GUI log; and to set or unset password. The Submit window will set and save job-related options.
- Use a launching capability to submit a job request periodically, at specific or repeating intervals.
- Use a Status window to verify the status of requests and FTA file transfers. Use the Status window also to delete a request, to send a specified signal to a request, to get a detailed status of a request, and to set or unset password.
- Obtain context-sensitive help by gliding the mouse cursor over a menu or field name in a window. A brief description of the menu or field will appear at the bottom of the display.
- Use the Load window to view a continually, updated display of the system load for NQE servers in the complex. This data may be grouped by host or by type of data. Users may also view data about a specific NQE server.
- Use the Config window to set specific user preferences and see how variables are currently set.

3.4 Cluster-Wide Job Rerun

If a job has been assigned to an NQE server and that server has been down for a specified length of time, NQE 3.0 provides support for rerunning that job request on a different NQE server in the cluster.

3.5 Interactive Load Balancing

The new ilb feature executes commands on an an NQE server chosen by the NLB. To use this, enter the ilb command followed by the command to execute. The NLB is queried to determine the machine to execute the command. The command is executed and I/O is connected to the terminal session or to an optional pipe to another command.

3.6 FTA Enhancements

FTA has been enhanced in NQE 3.0 to provide byte-level recovery of interrupted file transfers, to display additional data from the NLB, and to display the status of FTA file transfers in the `nqe` GUI.

3.7 Extensible Collector

It is now possible to store dynamic, site-specific information in the NLB database. This information is periodically sent to the NLB by each collector process, along with the data that is normally stored and updated in the NLB. Once the customized data is in the NLB, it can be used in policies or displays just as any other NLB data. NQE collects and stores the customized data, but the site defines, generates, and updates it.

The new `ccollect -C` file name option and the new NQE_CUSTOM_FILE_LIST variable provide the support for this feature.

3.8 Application Program Interfaces

NQE provides a language callable interface to `cqsub`, `cqdel`, and `cqstatl`. Sites can use these interfaces to provide customized commands for their users.

4 NQE Architecture

NQE consists of clients and servers. The Master Server is the NQE server with the consolidated information about workload and system status in the Network Queueing Environment.

The NQE Clients provide all the necessary commands for users to submit jobs and monitor the status of their work.

The Master Server matches the incoming workload and priorities to the system load of NQE servers in the cluster, and schedules jobs for the most appropriate NQE server. The selected NQE server may be the Master Server or an Execution Server. Jobs may arrive from other NQE servers as well as from Clients.

There may be only one NQE Master Server in an NQE cluster. The NQE Master Server runs the Network Load Balancer (NLB), the NQE SQL database, and the NQE scheduler. The NLB may be duplicated on other NQE servers, which may serve as a backup to the NLB if it becomes unavailable. The Master Server contains both client and server code, and may also function as a Client or an Execution Server as needed.

NQE Execution Servers collect information about system load and job status and send this information to the NLB on the Master Server (as well as to any backup Master Servers). Execution Servers use NQS, the Network Queueing System, to initiate job requests and FTA for file transfers. Execution Servers also contain client commands and may function as a Client as needed.

5 NQE Open Scheduling

The crux of workload management is scheduling. Many customer requests for new features revolve around scheduling needs. Scheduling needs for different sites differ and conflict, to the extent that Cray Research cannot provide for all these needs in a timely fashion. Therefore, the solution is to provide an architecture that allows for easy customization by the individual site. Additional priorities in developing the new model are the need to improve cluster support and the need to create a framework for jobs with parallel execution.

The solution is NQE Open Scheduling. The NQE scheduler is based on an SQL database and on Tcl (Tool Command Language). The use of an SQL database provides a standard interface for examining information in the database. Tcl is a scripting language similar to other UNIX shell languages, such as the Bourne Shell (sh), the C Shell (csh), the Korn Shell (ksh) and Perl. In particular, it provides an extension language to configure and customize applications.[1]

[1] Brent Welch, Practical Programming in Tcl and Tk, Prentice Hall PTR, Upper Saddle River, New Jersey, 1995, p.xxx.

By using Tcl, NQE 3.0 provides a mechanism for sites to write custom schedulers. There is a testing mechanism that can be used to test a scheduler before putting it into production. The model allows the choice of scheduler to be updated dynamically at any time. This simplifies changing schedulers for nights, weekends, or holidays.

The new model holds jobs requests in a central location until they are ready to be initiated. This allows system load to be examined at the time the job request is ready to execute.

The new model is layered on top of NQS. This preserves the current NQS environment and investment. It provides a migration path that allows sites to continue in the old environment while experimenting with the new one. Workload can be gradually migrated to the new environment as needed and as work schedules allow.

6 NQE Distributed Client/Server Model

The new NQE distributed client/server model consists of the NQE scheduler, the NQE SQL database, a lightweight server (LWS) paired with NQS on each Execution Server in the cluster and the NQE monitor.

6.1 NQE SQL database

The NQE SQL database stores information on the global configuration as well as on all system tasks and job tasks. All operations are done by updating information in the database and sending an event to the appropriate system task to read the information. Information is stored in system task objects or job task objects.

For example, all task objects have an attribute called state. There is one system task object for each LWS task (process). The state of each LWS system task object tells NQE if NQS is running on the corresponding execution server. Figure 1 shows an example of summary information about the LWS system task objects in the NQE SQL database.

```
% nqedbmgr select lws
  ID      CLASS  NAME        STATE
  s1        lws  irix      Running
  s2        lws  unicos    Running
  s3        lws  unicosmk  Running
  s4        lws  sunos     Running
  s5        lws  solaris   Running
  s6        lws  hp-ux     Running
  s7        lws  alpha     Running
  s8        lws  aix       Running
  s9        lws  red        Exited
%
```

Figure 1: NQE SQL Database System Task Objects

6.2 Scheduler

The NQE scheduler examines each job request and sends it to an Execution Server. It does this by assigning the job task object to an LWS.

6.3 LWS

There is one LWS system task object and process for each Execution Server. The LWS runs on the Execution Server and

performs remote SQL commands to the NQE SQL database to learn which job requests have been assigned to it. The LWS authenticates the user and then submits the job request to NQS.

6.4 Monitor

The NQE monitor checks the heartbeat updates of each of the system tasks to verify they are still running. If, for example, an Execution Server goes down, the NQE monitor will notice that its LWS has not reported in, and will mark the state of that task as Exited. The monitor will also delete old job tasks objects after the configured period of time has expired.

7 System Flow

Job requests that are destined for the NQE SQL database and scheduler are submitted directly to it (`cqsub -d nqedb`). When a job request first arrives at the NQE SQL database, a job task object is created to store all information about the request.

The job task object is given a state of New and assigned to the NQE scheduler. The NQE scheduler accepts the job request by giving it a state of Pending. Job requests will be in the Pending state until they are scheduled to run and assigned to an LWS (light-weight server). When this happens, the job request will have a state of Scheduled. The LWS on the selected NQE server will hand the request to NQS to initiate the job and change the state of the task object to Submitted.

When the job finishes execution, the LWS will give the task object a state of Completed and re-assign the task back to the NQE scheduler. The NQE scheduler will then assign it to the NQE monitor for retention or disposal of the database object. This has the advantage of keeping information about the job available for a specified period of time after the job has completed. The initial configuration is to keep this information for 24 hours after the task is assigned to the NQE monitor.

If the job fails authentication, it is given a state of Failed. If the job is aborted with a signal from `cqdel`, it is given a state of Terminated. In all cases, the LWS assigns the task object back to the NQE scheduler to determine the next step.

8 Operation

To add a test scheduler to a running system, 1) create the new system task object in the NQE SQL database, 2) update the object with the location of the Tcl file containing the scheduler functions, and 3) start the scheduler task. An example of these commands is:

```
nqedbmgr
% create systask scheduler.new
% update scheduler.new {s.schedfile new_sched.tcl}
% nqedbmgr start scheduler.new
```

In this example, the test scheduler will run simultaneously with the main scheduler, `scheduler.main`. To submit a job to the new test scheduler, use the attribute option to indicate the desired scheduler:

```
cqsub -la scheduler=new
```

9 Implementation of a Scheduler

The following NQE-defined scheduler functions, written in Tcl, may be revised by the local administrator:

- Tinit

  The Tinit function is called to initialize the scheduler. It will be called when the scheduler initially starts up or when it receives the TINIT event which may be generated with the nqedbmgr command:

  ```
  nqedbmgr ask scheduler.main tinit
  ```

  Use the Tinit function to initialize any needed variables.

- Tpass

  The Tpass function is called to select the next job to process and assign it to an LWS.

  The Tpass function is called on startup, on receipt of a TINIT event, and after any call to the NQE function sched_post_tpass.

- Tnewowner

  The Tnewowner function is called by the LWS and the monitor to increment the number of jobs requests under their ownership.

- Tentry

  The Tentry function is called when the scheduler starts up, when a new job request is submitted, when a job request completes, and when a job request fails authentication on a lightweight server.

  Based on the status of the selected job, this function assigns it to a lightweight server for hand-off to NQS or to the monitor to archive and/or dispose of the job task object describing the job request.

  Use this function to look for Pending, Failed, Aborted, Terminated job requests.

10 Applications

There are many possibilities for customer NQE schedulers. Many sites have scheduling requirements that are unique to their site. This section describes a few examples of custom scheduling applications that can be implemented using the NQE 3.0 scheduling feature.

10.1 CEO Scheduler

There are occasions when the normal scheduling needs to be bypassed and the jobs for a particular user needs to be executed immediately. This needs to be done regardless of the current priorities, system utilization, or policies. Often this is for a specific userid. The NQE scheduler can be altered to check for special userids and pass those jobs to the light weight server (LWS) for immediate execution. This can be done by checking the jobs userid against a list of high priority userids.

10.2 Enterprise Work Scheduler

Cray systems are often acquired to run a particular application that is essential to the success of the enterprise. These applications use most of the resources available from a Cray system. However, there can be idle time available and NQE is used to schedule additional work to benefit from the performance of a Cray system. It is still essential that the key application runs and gets the turnaround it needs. The NQE scheduler can be altered to check for "Enterprise" work. User defined attributes can be added when the jobs are submitted and these attributes can be examined in the scheduler and used as a sorting criteria when ordering jobs for execution.

10.3 Execution Server Shutdown

NQE environments can be complex and encompass a large number of execution servers. Occasionally one or more of these execution servers needs to be taken out of service for hardware or software maintenance. Often this is a planned and scheduled activity. If the dates and times of the scheduled outages are known, NQE could be altered so that jobs that run past the time of the scheduled outage would not be sent to that execution server. This can be done by checking the CPU time requested by the job and testing to see if the ending time would be past the time of the scheduled outage.

10.4 Historical Accounting Scheduler

NQE records accounting data about the work done on an execution server. UNICOS systems have extensive accounting data available for use and reporting. Large-grained system sharing can be done by summarizing previous days or weeks system usage from accounting data. This data can be summarized by userid or account id. By sorting usage, an ordered list of user priorities can be set on a regular basis. The NQE scheduler can be altered to read the summary file and use it as a sorting criteria for job ordering.

11 Summary

NQE 3.0 provides significant new functionality. In particular it addresses key requirements for scheduling and clusters. NQE 3.0 is the base for a new architecture that will be used to transition from the traditional NQS model to a distributed client/server model. This model is based on a client/server SQL database.

NQE 3.0 comes with scheduling that provides NQE 2.0 functionality plus a central job database. This central database holds jobs till they are ready to execute. At the time of execution, jobs are forwarded to an execution server to run. NQE 3.0 provides a script language for writing custom schedulers. There is a testing mechanism that can be used to test a scheduler before putting it in production. NQE 3.0 schedulers can be updated at any time. This simplifies changing schedulers for nights, weekends or holidays.

NQE 3.0 preserves the investment made on-site with the current NQS and NQE 2.0 structure. NQE 3.0 is layered on top and preserves the current NQS and NQE 2.0 environment while providing significant new functionality. The NQE 3.0 architecture will be the basis for parallel job scheduling and parallel job management. Over time all components of NQS and the current architecture will be replaced. As new functionality is implemented, the NQS presence will subside.

For a tutorial on writing NQE schedulers, visit Cray on the World-Wide Web at http://www.cray.com.