J90 cluster scheduling using NQE 3

Eric Lebeau, RENAULT Automobiles, 92500 Rueil-Malmaison, FRANCE
Phone: (33) 1 47774854, Fax: (33) 1 47773459, Email: eric.lebeau@ctr.renault.fr

ABSTRACT:*RENAULT's scientific computing center is made of several specialized servers, running different batch management products. Mid 96, RENAULT decided to buy a second crash simulation machine to be added to the existing CRAY J916 system ; this system turned out to be a CRAY J932. As we wanted the two machines to operate in cluster mode and in an attempt to unify the HPC center's batch environment, we decided to look for a new batch scheduling product. The NQE 3 environment was chosen as it could meet our requirements, but with lots of developements. The J90 cluster was brought into operation in December 96 and works fine. However, some questions have to be answered before this solution can be extended to the whole computing center.*

1 The HPC facilities at RENAULT

1.1 The scientific computing architecture

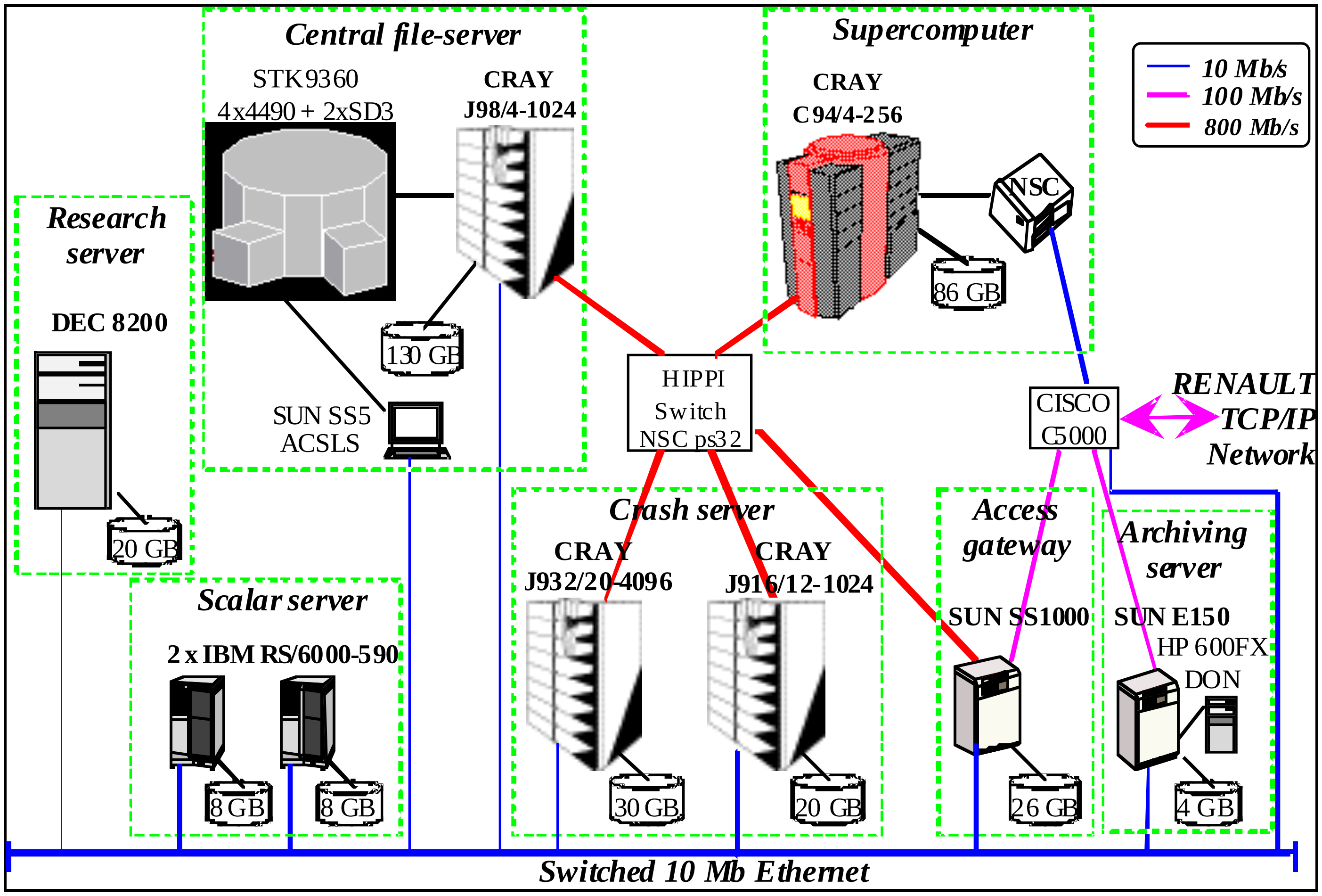
RENAULT's scientific computing architecture is based on central shared HPC resources located in a single computing center ; engineers are equipped with workstations (mainly SGI and SUN) on a one to one basis. All graphical interactive tasks, like meshing and data visualization, are done on these workstations. Simple simulation programs are also run on the user workstation, while all time-consuming intensive calculations are submitted in batch mode to the central computers.

1.2 Simulations made

The design of a car calls for many different kinds of simulations, among which:

. static and dynamic structure analysis
. crash (against a barrier, car to car ...)
. metal stamping
. kinematics
. engine cooling
. external aerodynamics
. engine internal flows
. combustion
. electro-magnetic compatibility
...

All complex simulations are made with commercial



applications like NASTRAN, ABAQUS, RADIOSS, STAR_CD, FLUENT ...

1.3 RENAULT's HPC center

The increasing demand for high performance computing resources has induced RENAULT's HPC architecture to evolve from an all-purpose big supercomputer to specialized computer servers, in order to achieve the best price/performance ratio depending on the application profile:

. vectorized and i/o intensive:	CRAY C90
. vector parallel:	CRAY J90
. scalar sequential:	IBM RS6000
	DEC 8200
. scalar parallel:	to be decided

A dedicated J98 and its WolfCreek STORAGETEK silo with 3490 and SD3 drivers under control of DMF acts as central file-server for restart files and NASTRAN modal bases.

An archiving service is also provided to the end users.

2 The "unified batch management" project

2.1 The batch environment

The HPC facilities are accessed in batch mode, except for the DEC server which is dedicated to research projects, and run 24h/24, 7d/7 unattended during nights and week-ends.

At the beginning of 1996, we had two batch products to manage our systems:

. CRAY/NQS for the C94 and the J916, with network access through NQE-R (subset of NQE 1 equivalent to the former RQS product)

. IBM/LOADLEVELER for the IBM cluster, introduced for its central queuing and load balancing capabilities.

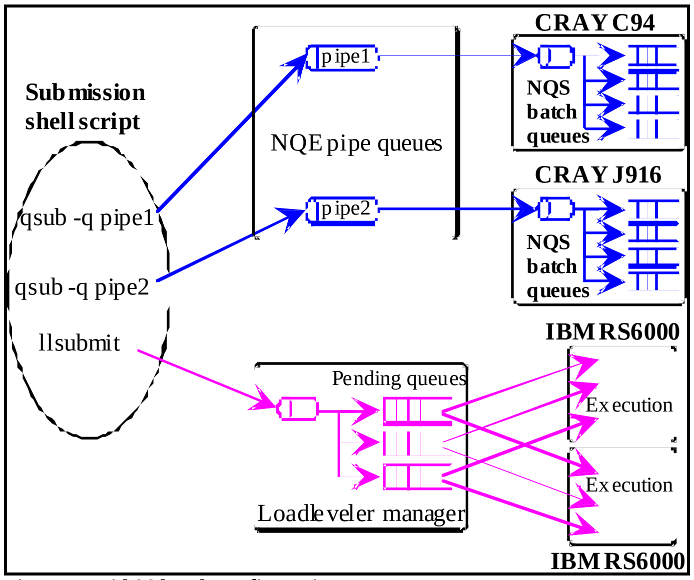


Figure 1: mid 96 batch configuration

All batch commands have been packaged with shell scripts in such a way that this disparity was transparent to the user:

. batch submission: static routing to one system or another depending on the application specified, NQS to LOADLEVELER resource definition and options transcoding, default job resources attribution:

. batch deletion: sending the proper order to the proper system

. job status: a single job status display is built every minute and can be accessed via a simple graphical interface with filter capabilities

. job outputs: all outputs return to the same place and have similar names

2.2 The project objectives

In spring 96, we decided to start a project on the batch management topic ; the main objectives were:

. to unify our batch environment on one single product, so as to get rid of all the developments previously described and to enhance user functionalities by taking advantage of the native client interface of the batch product

. to prepare to bring into operation the second crash simulation machine which was going to be added to the existing CRAY J916 before the end of 1996.

2.3 Functional specifications

First, we established a list of specifications for the "ideal" batch management system, among which:

. support of all major HPC platforms

. basic batch functionalities: ability to define several queues, resources limits enforcement (cpu, memory, disk ...), job tracking, including a detailled status of running jobs (cpu and memory actually used ...)

. centralized queuing so as to achieve efficient load balancing between different execution servers

. explicit dependency between submitted jobs

. advanced static routing: customizable algorithm for routing a job to a set of queues depending on the resources needed (including site specific resources like an application)

. fair-share batch scheduling: take into account at least the time in queue and the past resources usage of the user when ordering jobs within a queue (not simple FIFO !)

. continuous operation when an execution server in a cluster fails: all jobs should be requeued and run on the remaining servers

. resources conversion depending on the execution server (especially for the cpu unit)

. scheduling master server and submission server securization

. customizable graphical and line-mode user interfaces

2.4 Products short list

At the time the project started, we didn't know what the second crash machine was going to be, but we already knew that the batch product would have to be implemented on a UNICOS system : this constraint reduced the short list to 2 products : NQE 3 from CRAY and LSF from Platform Computing.

The study of NQE 3 product specifications and the on-site test showed it didn't perform many required functionalities ; a further study with CRAY France demonstrated that almost everything we needed could be implemented in the product using the new database mode and with some developments.

The tests of LSF planned in the summer of 1996 didn't go far as we had some difficulties in obtaining the UNICOS client from our french distributor and were not able to make it work correctly, neither in native mode nor in NQS interoperability mode (we eventually found out that it had not been ported to UNICOS 9 at this moment).

The final choice for the second crash machine was made in september for a CRAY J932, and we decided to bring the cluster into production with NQE 3, all developments beeing subcontracted to CRAY France.

3 J90s clusterization with NQE 3

3.1 NQE 3 technical architecture

Version 3 of NQE implements a very important new feature: the database mechanism. With the previous versions of NQE, jobs were routed at submission time to an execution server ; the NLB provided a way to choose the "best" destination base on the server's load, but job queues were on the execution server itself, in a classical NQS mode.

When using the new database mecanism, jobs are inserted in the base at submission time ; a program, the scheduler, examines the database at regular intervals to check if jobs may be executed.

When the scheduler decides that a job may be executed, it sends the job to a process named LWS on the chosen execution server. The LWS interacts with NQS in order to controls job execution : it dispatches the job to NQS, returns the job status to the central database, and takes care of special events like job signalization/deletion or LWS shutdown.

The default mode for sending jobs to NQS is immediate execution (qmgr schedule now), which means that NQS's function is mainly to enforce resources limits. It is also possible to allow queuing on the execution server by changing a parameter on the LWS. We have not tested this possibility.

Another process on the execution server, the collector, periodically sends back to the NLB server detailled information about running jobs (cpu and memory used ...) and global load information ; it can be customized to collect whatever information is needed.

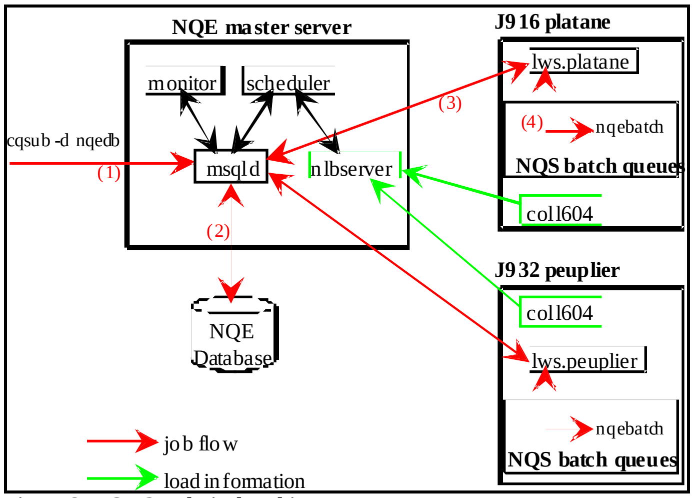


Figure 2: NQE 3 technical architecture

NQE 3 also provides a set of graphical user commands for job submission and tracking. This environment is very open: the scheduler and the LWS are written in tcl, which is an interpreted command langage, and the client programs are written in tk/tcl, a graphical extension to tcl. The collector can be customized by creating new objects that the site wants to monitor, and the algorithm used by the NLB to give a list a suitable LWS for a job may be defined by the administration through custom NLB policies.

3.2 Implementation at RENAULT

NQE 3 comes with a very simple default scheduler only capable of sending jobs to execution servers in a round-robin fashion. Many of the functionality that we absolutely needed to start the J932 in cluster mode with the existing J916 had to be implemented in the scheduler. We describe the main ones.

3.2.1. Queues and global constraints

Having a central queuing mecanism in NQE 3 implies that all treatments related to queues and constraints must be done by the scheduler. Queue allocation is done at submission time depending on the cpu and memory resources needed by the job ; the job "enters" the first queue that fits its requirements or falls in a "Failed" state if none was found. At every pass, the scheduler checks all contrainsts to see if a job can be started. As every constraint must be manually coded in the scheduler, we were content with implementing only the minimum we needed : queue run limit per LWS, global and per user run limit per LWS.

3.2.2. Pipe queues

The J90 cluster runs both parallel and sequential applications, mixes very long simulations with short preparation jobs, and is shared by two groups of users. As we didn't want to multiply the number of queues, we decided to implement a pipe queue mechanism. The user may specify a pipe queue when submitting the job using a special argument "-la pipe=xxxx" ; for each pipe queue argument, there is an array of eligible destination queues to which the scheduler restrains its choice when the job enters the database.

3.2.3. Job dependency

The so-called job dependency mecanism implemented in NQE did not satisfy our need because either the dependency is checked before job submission and then we loose reliability (will the user workstation be able to submit the job when time comes) and equity (the time in queue is accounted for only when the job is actually submitted), or the job has to be running to check if it may start (!). That's why we implemented real interdependency between submitted jobs : the user may indicate using an argument "-la waitfor=xxxx" that this job can't be started before the specified job has finished. The checking is done by the scheduler during the scheduling pass.

3.2.4. Fair-share batch scheduling

Users were very satisfied when CRAY implemented the fair-share batch scheduling in NQS because it broke the static FIFO logic which allowed a single user to monopolize resources by simply submitting many jobs at a time and replaced it with a fairer system taking into account the time in queue and also the past usage of the resources by the user. We had to have the same kind of dynamic queue reordering with NQE 3, and it was implemented in the scheduler : at each scheduling pass, the scheduler reorders the jobs in each queue using the same algorithm as NQS for priority and usage decrementing computations. Note that it only takes into account the cpu used by finished jobs because this information is documented by the default NQE mecanisms.

3.2.5. Job priorization

Another very usefull feature was also implemented in the scheduler: the ability for the administrator to prioritize a job by putting it at the top of the queue (equivalent to the "qmgr schedule first" command) or by forcing immediate execution on a given LWS (equivalent to the "qmgr schedule now" command).

3.2.6. Central hold/release

A central hold/release had to be implemented in the scheduler so that the administrator can force pending jobs to be held in queue and running jobs to be checkpointed (an the associated release commands) ; unfortunately, there is no "hold" state in NQE 3: this state had to be implemented using a special scheduler variable and cannot be seen by the user. Note also that this function implied some development in the LWS program as it is the LWS that communicates with NQS and asks for hold/release. Nevertheless, if a job is held directly on the execution server using a NQS "qmgr hold request" command, then the scheduler is not aware of it and the job is still considered as running.

3.2.7. Job tracking

An external command written in tcl queries the NLB and the database every minute and constructs a NQS-like detailled display of jobs including cpu and memory required/used and site-specific informations like the queue and job dependency.

3.3 The missing functionalities

3.3.1. Administration interface

Some administration commands are still missing in our implementation, especially regarding dynamic queue definition and modification ; defining a new queue implies a modification in the scheduler and, of course, a restart.

3.3.2. Continuous operation when a server fails

NQE continuously monitors its execution servers and provides a mecanism by which when an execution server fails (customizable timeout), all jobs running on this LWS are requeued in the database (ie change state from "submitted" to "pending") and are eligible for running on another LWS. We have not tested this functionality, but it seems that the problem of canceling these jobs when the LWS restarts has not been properly dealt with.

3.3.3. Master server protection

It is not possible at present time to secure the NQE database ; therefore, the only way of securing the NQE master server is to have a fault tolerant machine configuration.

3.3.4. NLB policies

In our configuration ,batch queues are statically defined in the scheduler in a NQS-like way ; Nevertheless, everything is ready for using NLB load informations in order to route jobs to the less loaded system, or even to dynamically adjust queues run limit depending on execution servers loads.

3.4 User experience

The J90 cluster was brought into production using NQE 3 on December 2nd. Our configuration is made up of 3 pipe queues and 7 batch queues ; all jobs are run in a single NQS batch queue and there is no queuing on the execution server. We run an average of 4000 jobs/month.

In the first months of operation, we experienced 2 problems:

. a bug in the product that caused the NQE environment to grow in memory on the master server (SUN SS1000) and on the execution servers ; we had to restart the scheduler every week to free memory and avoid memory shortage on the master server (nevertheless, it happened once, causing the scheduler to abort. This bug is now fixed by CRAY.

. the job tracking command was very slow (over 50 seconds) and very cpu-time consuming ; the problem was fixed by changing the way this program queried the NQE database, and the command now takes about 10 seconds.

There still seems to be a remaining problem on the hold/release mecanism (which has been recently brought into production) that causes the scheduler to loop in some special circumstances ; this bug happened twice on our site and is being analysed by CRAY France.

Apart from that, we had no major trouble in 6 months which proves that NQE 3 is a very robust and reliable environment.

4 Conclusion

NQE 3 is a very robust and reliable product ; the J90 clustering project demonstrated that it is very open, since many functionalities we needed could be implemented without great difficulty.

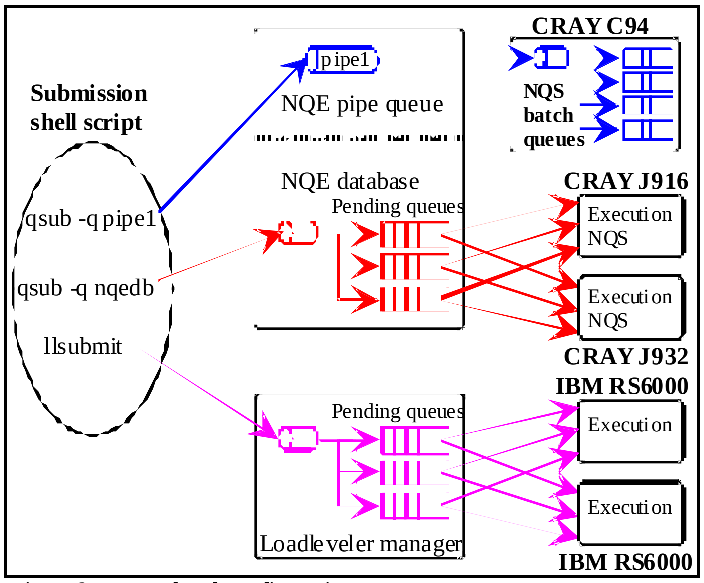


Figure 3: present batch configuration

However, we have not decided yet to extend this environment to the whole computer base for different reasons.

4.1 Product scalability

We are concerned about the product's behaviour if the activity grows too much ; the HPC center runs more than 20000 jobs/month, which is at least 5 times more than the J90 cluster activity :

. the scheduler is written in interpreted tcl and must examine all jobs in the database at each scheduling pass,

. the database itself will significantly grow, and database queries may become very long, with unknown possible consequences on NQE

There is another uncertainty about the load induced by NQC queries to the database: about 100 users would simultaneously want to have a status display, refreshed every minute or so, to monitor their jobs. We today avoid this problem by constructing every minute a single image of the jobs submitted and put it into a file that is accessed by users commands via NFS. But if we want to take advantage of the capabilities of the NQC native interface, all these requests will be direct queries to the NQE database.

4.2 Scheduler complexity

The scheduler may become a very complex program if we try to port all the NQS constraints capabilities which are used in an all-purpose batch configuration like the one we have on the CRAY C94. As a matter of fact, for the J90 cluster project we only implemented a few batch the queues with basic constraints, and it still required about 2000 lines of development. But NQS also offers constraints on memory, disk space and PEs, and the additional level of "complexes" (constraints on a group of queues). Coding all these constraints and all connected administration commands seems very complicated and would bring about reliability and probably performance problems.

4.3 Support and maintenance

Last but not least, given the extent of developments needed to build a NQE solution using the database mecanism, we face the problem of the maintenance strategy : should the customer develop and maintain its solution, or should CRAY support teams take on this task, as for us ? And in this case, how can we be sure that we will have an efficient support on this strategic service for a HPC center ?