Automatic Operations on Y-MP and T3D Systems

Denis Lubin, Pascal Richard, and *Elizabeth Charon,* Centre d'Etudes de Limeil-Valenton, Commissariat à l'Energie Atomique, Villeneuve-Saint-Georges, France

Abstract

The Cray Y-MPs and T3D running at CEL-V and CEV-M offer a 24 hours a day, 7 days a week service. Due to shift work changes, the systems need to be autonomous during nights and week-ends. This paper describes how we manage the automatic batch submission by a system using Fair Share Scheduler and NQS 8.0 possibilities. We compare the autonomous functioning with the former human control and present our ideas in general for a fair and optimized job scheduling automation on Cray computers.

Introduction

The CEA "Commissariat à l'Energie Atomique" is made up of 12 research centers; in two of them, CEL-V and CEV-M, located near Paris, hundreds of engineers work on research in nuclear physics using numerical simulations which require high-performance computers. Consequently, these centers host 4 powerful Cray machines :

- 3 "PVP": 2 *Y-MP 8/128* and 1 *Y-MP 4I/128*,
- 1 "MPP": a 64 nodes T3D with 128 processors and 8 Mwords/processor.

1 Requirements

Our CRAY computers provide a non-stop service. Their power is shared between different "departments" on a "CPU quota" base. The operating mode is different during *prime-time* (working days) and during nights and week-ends.

1.1 During working hours

During working hours, the CRAYs are available for interactive and batch work, but only for short runs (up to 10 CPU minutes), in order to offer the sharing of the machine between a large number of users.

Batch mode is mainly used to prepare and test the jobs which will be executed during nights and week-ends.

Prime time needs are basic: a good response time for interactive sessions, a good throughput for batch jobs and a fair repartition of the resources between users.

1.2 During night and week-end

Nights and week-ends are devoted to jobs which require important CPU and memory. Jobs are executed according to the following rules:

- **Intra-department scheduling**

  The users do not submit their jobs by their own. Each department provides us with an ordered list of jobs to submit.

- **Departments CPU quotas**

  The CPU quotas granted to departments determine priority in the choice for the jobs to be executed.

- **Restricted job dependency**

  Two jobs are said dependent if the second one must wait for the completion of the first one before starting execution. As job dependency is not controlled by NQS, we traditionally assume in our centers that job dependency exists between jobs of same *job name* and same *user name*.

2 Former organization

Before 1995, the computer center was staffed 24 hours a day by operators whose work was different during prime-time (users working hours) and non prime-time (night and week-ends).

2.1 Prime-time work

During the day, the operators supervise computing center equipments and provide help to users.

2.2 Night and week-end work

During night and week-end, the operators submit jobs mentioned on the departments lists through a specific interface. They control the number of running jobs, pay attention to job dependencies and consult home-made *counters* to select jobs of different departments.

The *counters*, as shown in figure 1, display the quotas granted to departments, and the remaining time.

```
date      : 17/02/94 16:47:02
planning  : 17/02/94 16:46:51 (12h period)

dep. %    cpu/period      daily          weekly
          used  remain    finis.run.     finished
d1 (29)   00:00 23:24     03:32 00:00    031:10 (17)
d2 (24)   00:01 02:21     03:16 00:01    073:34 (42)
d3 (16)   00:00 11:00     00:00 00:00    001:56 (01)
d4 (15)   00:02 00:54     07:22 00:02    061:12 (35)
d5 (04)   00:00 02:34     00:17 00:00    001:17 (00)
d6 (10)   00:00 06:23     00:00 00:00    005:38 (03)
d7 (01)   00:00 00:37     00:00 00:00    000:00 (00)
d8 (01)   00:00 00:37     00:00 00:00    000:00 (00)
```

Figure 1: home-made counters display

3 New organization

As operators are now present only from 6am to 9pm, the organization of the computing center has changed:

From 8:30 am to 5 pm, operators are doing the same work as in the former organization.

From 5 pm to 9 pm, operators submit all the jobs. They can solve some jobs problems as network failures, bad `qsub` options, etc...

From 9 pm to 6 am, machines are working in the new automatic mode.

From 6 am to 8:30 am, operators make a report on night operations, address it to each department, and switch to day mode.

4 CPU resource partitioning

Before the arrival of the T3D, each user was free to use as much CPU as he wanted during working hours. Departments were penalized during night proportionally to day usage (cf "counters").

When the T3D was connected to the Y-MP 4I, some users moved to T3D code development. As T3D compilations need a lot of Y-MP CPU, we decided to partition the CPU resource between "classic" Y-MP users and T3D users.

To achieve this partitioning, we used the *Fair Share Scheduler* [4, 5] and defined *resource groups* as shown in figure 2.

The third group (system administrators) can use 20 % of the power but actually uses about 2 % of the resource.

As the Fair Share Scheduler accounts for the CPU used by each resource group, we defined a resource group for each of our departments and abandoned our home-made counters.

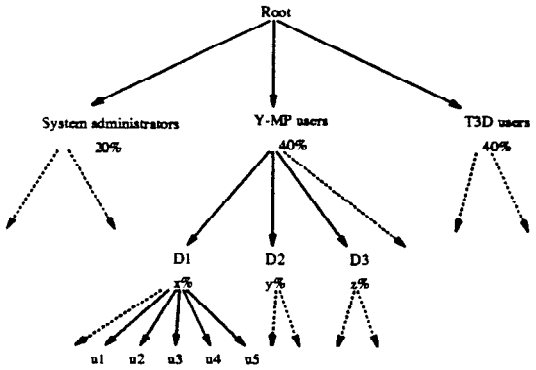

Figure 2: Fair Share groups arborescence for the Y-MP 4I (T3D host)

5 Control automation

As machines have to be autonomous from 9pm to 6am, we studied how to satisfy automatically the requirements stated in § 1.

To achieve the automatic control, we chose an improvement of our usage of *NQS* rather than the design of a new large product. Usage of *URM* [4, 5, 3] was turn down as the problem to solve was a "pure batch" scheduling problem. Usage of *NQS user-exits* [4] and *Fair Share Scheduler* was a key tool in the development of the product.

The *Fair Share Scheduler* provides us the accumulated CPU usage of each user and resource group (our departments).

NQS user-exits are site-custom C functions linked with `nqsdaemon`, the process managing NQS requests. Those functions are called at strategic points by standard `nqsdaemon` functions and can access all C structures of the program.

The two user-exits we customized are :

- `nqs_uex_jobselect.c`, provided to customize the job selection policy,
- `nqs_uex_shrpri.c`, provided to customize the *NQS fair share priority* calculation.

5.1 Job Ordering

All the jobs for the night wait in a unique batch queue. We have to start them according to the departments CPU quotas without changing the order imposed by each department. Our job scheduling is done by NQS *intra queue priority weighting factors*. Those factors are:

- Time-in-queue (the only non null factor in FIFO strategy),
- Requested CPU Time,
- Requested Memory,
- Fair Share Priority.

5.1.1 Departments CPU quotas

To comply with the departments CPU quotas, we abandoned our old system of counters for the fair-share scheduler counters because they are integrated in the system and easy to interface with NQS.

The default NQS fair-share priority is a per user priority. We customized the user-exit `nqs_uex_shrpri.c` to get the same priority for all users of a resource group (our departments). Based on an example provided by CRI, this modification was very cheap (2 lines changed).

5.1.2 Intra-department scheduling

The jobs of a department are submitted to NQS in descending priority order, following the department directives.

5.1.3 How to deal with those two criteria

The `qmgr` intra queue weighting factors we use are:

```
set shed_factor share  20
   # first sorting criterion
set sched_factor cpu    0
set sched_factor memory 0
set sched_factor time   1
   # second sorting criterion
```

5.2 Job dependency

To take account of the restricted dependencies described in § 1.2, we modified the user-exit *nqs_uex_jobselect.c*. Each candidate to execution is proposed by NQS to this user-exit which can force or refuse the execution. This modification costs about 50 lines.

5.3 Memory management

The jobs of our customers can request up to 100 Mwords of memory. As our Y-MPs have only 128 Mwords, we have to use some tricks to allow large jobs to run without leading to excessive memory swap.

Constants used in this subsection :

N_{proc}: number of processors ($N_{proc} = 4$ or 8 on our Y-MPs),

H_{max}: maximum available memory for "hog" processes ($H_{max} \approx 115 Mwords$ on our Y-MPs),

5.3.1 What we can do with standard NQS

The three parameters provided by NQS to optimize memory usage are :

1. the maximum number of running requests,
2. the maximum memory per request,
3. the maximum memory for all requests.

We set the *maximum number of running requests* to $N_{proc} + 2$ as it seems to be a good compromise to avoid CPU idle and allow large jobs to enter the memory.

We set the *maximum memory per request* to 100 Mwords.

The *maximum memory for all requests* is more difficult to tune as shown below:

- if we set the maximum memory to a low value

  ```
  set global memory_limit=115mw
     # 1 * Hmax
  ```

  There is no memory swap. However, large jobs can be blocked forever as each finishing small job is replaced by a small one; and late prime-time requests can be blocked in queue if a large job is in memory.

- if we set the maximum memory to a high value

  ```
  set global memory_limit=345mw
      # 3 * Hmax
  ```

  The scheduling of jobs is maintained, late prime-time requests can enter the memory. However, several very large jobs can be in execution simultaneously, generating a lot of swap activity which may block late interactive users sessions.

So we decided to modify a user-exit to control the NQS memory usage.

5.3.2 What we did with user-exits

Enforcing the job scheduling is very easy (1 line in `nqs_uex_jobselect.c`), but is not sufficient: if several jobs requesting each about 3/4 of the memory are at the head of the queue, only one of them can be in memory, so $N_{proc} - 1$ processors are idle and 1/4 of the memory is free.

Enforcing the maintain of large job scheduling and limiting the space occupied by those jobs insure the execution of large jobs, avoid swap between large jobs and limit swap activity.

We consider jobs requesting more than H_{max}/N_{proc} as *large jobs*. We limit the memory available for those large jobs to H_{max} and enforce their scheduling. The total memory limit (large + small) is fixed to $2H_{max}$.

This modification represents about 70 lines of C in `nqs_uex_jobselect.c`.

6 Prime-time improvements

We took advantage of the knowledge we gained automating the night submission to improve the day NQS configuration.

As in non prime-time, we have constraints to respect:

- control of job dependency (as in non prime-time),
- "fair" scheduling of requests,
- control of machine-gun submissions,
- control of swap (critical for interactive use).

The control of job dependency is realized using the same mechanism as in non prime-time.

6.1 "Fair" scheduling of requests

In prime-time, we autorize users to ask up to 10 CPU minutes and 32 (48 on Y-MP 4I) Mwords of memory per request. As we very often have more than 20 requests in queue, it is important to find a "Fair" priority policy.

So, we defined a batch queue **normal** which accepts requests asking for less than 10 minutes and 32 Mwords (48 Mwords on Y-MP 4-128) and set the intra-queue weighting factors to the following values:

```
set sched_factor share  0 # S=0
set sched_factor cpu    6 # C=6
set sched_factor memory 1 # M=1
set sched_factor time  10 # T=10
```

This setting favors requests asking for little CPU and little memory, but limits to

$$r_{max} = \frac{T}{T - S - C - M} \approx 3.33$$

the ratio of waiting time between two jobs.

To avoid very short jobs to wait, we defined a queue **express** accepting requests asking for less than 30 CPU seconds and 8 Mwords.

Requests asking for MPP resources are routed to the **pe128m10** batch queue which accepts requests asking for less than 10 MPP minutes on 128 PEs.

6.2 Control of machine-gun submissions

6.2.1 What we can do with standard NQS

To prevent a user from monopolizing the NQS resources, NQS offers the possibility to limit the number of running jobs per user:

```
set complex user_limit=2 prime-time
```

The weaknesses of this setting are:

- a single user (not using multitasking) cannot use all the processors of the machine,
- a "relentless" user could submit a sequence of jobs which will arrive at the same time at the head of the queue. Each finishing job of this user will be replaced by a job of the same user.

6.2.2 What we did with user exits

When a request is elected for execution, the date of arrival in queue of other requests from the same user is changed to the current date. This modification represents about 10 lines in **nqs_uex_jobselect.c**.

The behavior is the same as if users waited their job to enter execution to submit the next one. It provides us a fairer order and allows us to set the **user_limit** to the number of processors.

6.3 Control of swap

To deal with the swapping of interactive jobs, we changed some **nschedv(8)** parameters. We set **small proc** to 1000 clicks to accept **sh**, **csh**, **vi** and **emacs** as small processes, and **itime** to 120 to avoid those small processes to be swapped before 2 minutes without terminal input.

Other **nschedv** parameters can be usefull. Among them, the limitation of **hog_max_mem** and the usage of **nfactor_in** and **nfactor_out** are described in [1, 2].

Conclusions and future work

As there are no longer any operator during the night and week-end, we replaced the human control by a system based on NQS user-exits and Fair Share Scheduler counters. As shown in this paper, NQS user-exits are easy to use and well adapted to customize job management. The Cray Fair Share Scheduler is a reliable tool allowing us to select jobs according to departments quotas and to grant a specific percentage of CPU devoted to Y-MP frontal executions for the T3D.

The automatic organization has now been experimented for two months, offering the same service to users as before. We gain from the machines a quite full usage of their processors (97 %) as well as in the manual control.

We also acted on prime-time submission control, providing a fairer job scheduling during working hours.

We now think to various enhancements to complete the automation. Among others, we are working on:

- integration of general job dependency management,
- automatic action on network problems,
- automatic night report,
- better control of swap during prime-time.

References

[1] George W. Vanden Berghe. Insuring dedicated throughput for high priority work in an undedicated environment. In *CUG 1994 Fall Proceedings*, pages 361-370, 1994.

[2] Chris Brady, Don Thorp, and Jeff Pack. Priority-based memory scheduling. In *CUG 1994 Spring Proceedings*, pages 221–223, 1994.

[3] Hubert Bush. Unicos 8.0 experiences. In *CUG 1994 Spring Proceedings*, pages 237–238, 1994.

[4] Inc. Cray Research. *UNICOS System Administration, SG-2113 8.0.*

[5] Inc. Cray Research. *UNICOS Tuning Guide, SR-2099 8.0.*

[6] Joseph Harrar and Francis Belot. Live operations at cea-cel/v. In *CUG 1994 Fall Proceedings*, pages 213–221, 1994.

[7] Marko Lemieux. Throughput degradation analysis in a heterogeneous environment. In *CUG 1994 Fall Proceedings*, pages 395–401, 1994.