Job Scheduling Under the Portable Batch System

Robert L. Henderson[1]

NAS Systems Division
NASA Ames Research Center

Abstract

The typical batch queuing system schedules jobs for execution by a set of queue controls. The controls determine the queue from which jobs will be selected. Within each queue, jobs are typically selected in first-in, first-out (FIFO) order. This limits the set of scheduling policies available to a site.

The Portable Batch System (PBS) removes this limitation by providing an external scheduling module. This separate program has full knowledge of the available queued jobs, running jobs, and system resource usage. Sites are able to implement any policy expressible in one of several procedural languages. Policies may range from "best fit" to "fair share" to purely political. Scheduling decisions can be made over the full set of jobs regardless of queue or order. The scheduling policy can be changed to fit a wide variety of computing environments and scheduling goals. This is demonstrated by the use of PBS on an IBM SP-2 system at NASA Ames.

1: Introduction

Job scheduling consists of two activities. First is selection for execution of a job or jobs from the set of submitted jobs. Second is the allocation of memory and CPU resources among the set of jobs that have be selected for execution. The second activity is in the domain of the operating system kernel and will not be discussed in this paper. The first activity is typically part of a batch add-on subsystem and is the subject of this paper.

The typical batch subsystem is queue-based and schedules or selects jobs based on a set of queue controls. The nature and availability of these controls often limit the range of policies a site may implement. The scheduling policy at a given site may be a computer science related problem, such as determining the best fit of jobs in memory or aiming toward maximum CPU usage. Other sites however have policies that are driven by factors that are not related to computer science. These factors frequently include finance and even office politics. For example, priority may be given to jobs submitted by the department that owns the hardware, to a user who has "money" banked in his/her account, or to a researcher whose project is in favor with the management.

---

1. Computer Sciences Corporation, NASA Contract NAS 2-12961, Moffett Field, CA 94035-1000

To overcome policy limitations, the Numerical Aerodynamic Simulation (NAS) Facility at NASA Ames Research Center has developed a new batch system which provides an external job scheduler. This detachment of policy and implementation results in full freedom to express a site's job scheduling policy.

2: Queue Based Scheduling

A number of batch systems are available for open systems. The early ones only supported serial jobs and were commonly found on supercomputers. With the increasing interest of cluster computing, batch systems became more important and their capabilities have been expanded to include knowledge of parallel jobs.

It is interesting to examine several of these systems to see the common approach to scheduling and the weakness therein.

2.1: Early Batch Queuing

The first batch job queuing system which gained widespread use was the Network Queuing System (NQS) developed at NAS in the mid 1980s.[1] NQS supported multiple queues of several types. Pipe queues fed jobs into execution queues. Jobs in any given execution queue were placed into execution in a pure FIFO order. Limited controls were available to the systems operators and administrators. Queues could be turned on or off and the number of concurrent running jobs in each queue could be set. There was no provision to examine the level of system resource usage.

2.2: Current Batch Systems

Three newer batch systems are popular today. Each provides support for workstation clusters and improved scheduling controls. However, the controls still tend to be queue based.

2.2.1: Condor

Condor, a batch system designed to distribute serial jobs among workstations in a cluster, and developed at the University of Wisconsin, can be credited with starting the interest in cluster computing in 1987 and 1988[2] [3]. Support for PVM parallel jobs has been added. While Condor does not provide queues as do most other batch systems, its concept of classes serves much the same purpose.

When a job is submitted to Condor, its requirements are sent to a central negotiator daemon. System information, including load average and current owner keyboard/mouse activity, is forward to a collector daemon. The negotiator attempts to match jobs with available hosts. This is accomplished by taking the job requirements, expressed as logical expressions, and the system information and applying a set of rules. If the rules match, the job is initiated.

As a general batch system, two limitations exist with Condor as a result of its design goal to fill empty cycles on workstations. The rule set is limited to determining if a job can run somewhere. It does not provide the "best" ordering. Also, Condor provides no usage limit enforcement.

2.2.2: Distributed Queuing System

Developed at the Super Computations Research Institute at Florida State University, the Distributed Queuing System (DQS) has gained popularity both on large systems and workstation clusters.[4] Release 2 of DQS provided support for PVM. This support was removed in release 3 in favor of developing support for MPI.

DQS is typically set up with a number of queues. Jobs are automatically entered into one or more queues based on the job requirements. Each queue points to a host and has a specification of the load conditions under which jobs may be run on that host. The number of concurrent jobs from the queue can also be controlled.

2.2.3: Load Sharing Facility

The Load Sharing Facility (LSF) is a commercial product based on the Utopia project at the University of Toronto.[5] Unlike the other batch systems, LSF is designed to load level interactively run processes. It comes with an add-on batch subsystem called lsbatch. Lsbatch is queue-based. Jobs are placed into a queue based on a large number of parameters. Each queue may point to one or more hosts on which jobs may be executed. On each host, a load monitor reports current resource usage to a central collector. This information, together with more parameters including queue priority, load average setting, and time windows, is used to determine from which queues jobs can be initiated.

Of the various batch systems, lsbatch provides the widest assortment of controls. However, a site is still restricted to the provided controls.

3: The Portable Batch System

With much NQS experience on supercomputers, participation in the POSIX working group on Batch Extensions, and increasing interest in parallel processing, the NAS Division at NASA Ames found the scheduling capability of current batch systems lacking. Therefore, NAS decided to produce a second generation batch system which would support not only a range of system types, but also allow for the specification and implementation of almost any scheduling policy. This system is called the Portable Batch system (PBS).

3.1: General Description

The general appearance of PBS is similar to that of other batch systems with the major exception being scheduling. In addition to the usual collection of:

- client commands for submission, modification, and monitoring jobs,
- operator and administrator commands for configuration, modification, and monitoring the batch system,
- an API library for interfacing the commands to the Batch Server.

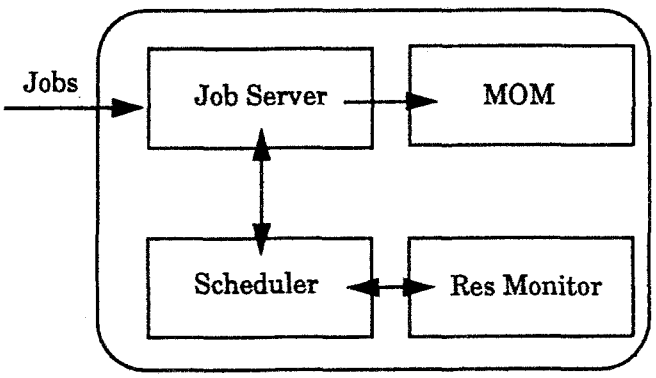
PBS comes with four daemon processes. Three are similar in purpose to daemon processes found in other batch systems. The fourth is the basis of the new capability. The four daemons are:

- The Job Server - owns and manages jobs and queues, is a central collection point for jobs, and a focal point for client communication. The server will also transfer jobs to the associated execution server or to other Job Servers.

The Resource Monitor - gathers resource availability and usage information about the host system on which it runs.
The Job Execution Server - is known as the Machine Orientated Miniserver (MOM), because the Execution server is the mother or shepherd of all batch jobs on a host. MOM places jobs into execution, monitors and controls their resources usage, and cleans up after they complete.

- The Scheduler - is a process which obtains information about jobs ready to run or currently running from the Batch Server and about resource availability and usage from the Resource Monitor. The scheduler then directs the server as to what, if any, action should be taken.

Figure 1. PBS on a Single Host



The functional distribution of services among the four daemons allows PBS to support a wide range of system types. For example, on traditional supercomputers, all four daemons may be (but are not required to be) on the one system as shown in Figure 1. Jobs arrive in the Job Server, and information is exchanged between the Scheduler and both the Server and the Resource Monitor. If a job is selected for execution by the Scheduler, it is sent to MOM to be executed.

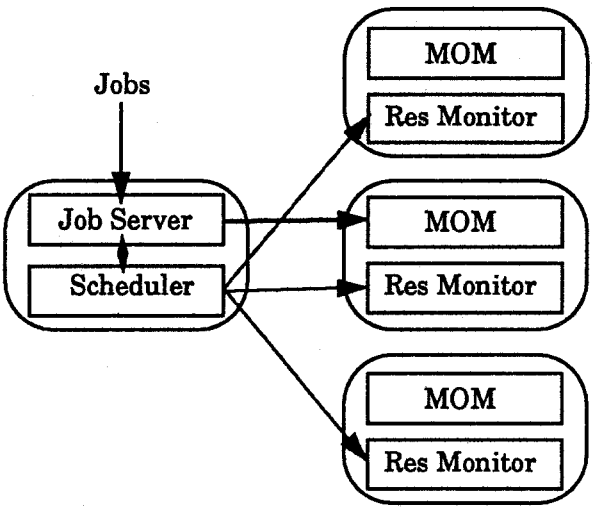
With a workstation cluster, the PBS daemons are arranged as shown in Figure 2. The flow of information is the same as in Figure 1 with the exception that there are multiple Resource Monitors for the Scheduler to talk with and multiple MOMs. The Scheduler tells the Server which MOM (host) should run the job.

3.2: External Scheduler

The power of PBS comes from the separation of functions into different processes as shown in Figures 1 and 2. This is especially true of the separation of the scheduling function from the other pieces of the system. This separation and the tools provided allow a site to implement any scheduling policy.

To appreciate the power of the Scheduler, it is necessary to have a basic understanding of the interactions between the scheduler and the other parts of the batch system, and the general flow within the scheduler.

Figure 2. PBS on a Cluster



3.2.1: Interface to the Batch System

The Scheduler is contacted by the Server when (1) a new job is queued, (2) a running job terminates, or (3) a timer expires. The Scheduler uses the same Applications Programming Interface (API) as do the user and administrator commands, to query and direct the Server. The Server grants the Scheduler full administrator privilege. Thus the Scheduler can run jobs, delete jobs, signal jobs, and even reconfigure the queues and the Server itself. The API routines commonly used by the Scheduler are:

- pbs_statjob — queries status of an identified job or all jobs in a queue or all jobs in the server.
- pbs_statque — queries status of one or all queues.
- pbs_statsvr — queries status of the server.
- pbs_runjob — runs a job at a specified host.
- pbs_deljob — deletes a job from the batch system. If the job is running it is killed.
- pbs_holdjob — places a hold on a job; on supported systems this causes a running job to be checkpointed and re-queued.
- pbs_rlsjob — releases a hold on a job.
- pbs_sigjob — signals a job; can be used to suspend a job.

3.2.2: Interface to System Resources

The Resource Monitor exists to provide the Scheduler with knowledge of the available system resources and of the resources currently being used. What resources are known to the Resource Monitor are specific to the host. Generally, the known resources are: total and available memory (real and/or virtual), swap space, load average, and file system size. Usage information about a specific job (session) is available on most systems. On workstations, the time since the last use of the keyboard or mouse is available.

In addition to the information available from the operating system, the Resource Monitor can be provided with a file containing "static" information such as availability of software or time of permitted access.

The Scheduler and the Resource Monitor use a special API to communicate. The Scheduler sends the Resource Monitor a complex query which is made of many *keyword* strings. The Resource Monitor responds with the value filled in for each string. For example, if the Scheduler sends:

```
pids[job=123]
totmem
phymem
loadave
```

the Resource Monitor might respond with:

```
pids[job=123]=456,457,459
totmem=33554432
phymem=16777216
loadave=1.5
```

informing the scheduler that the pids in session 123 are 456, 457, and 459, the total virtual memory is 32MB, the total physical memory is 16MB, and the load average is 1.5.

The information thus received from the Resource Monitor is available to the Scheduler for decision making.

3.2.3: General Scheduler Flow

The general flow during a scheduling cycle starts with the wake up call from the Server. The Scheduler queries the Server for information on jobs and optionally on queues and server attributes. The Scheduler also queries the Resource Monitor for information on resource availability.

Next, a site-supplied procedure evaluates the information and selects from a number of possible actions: (1) run one or more jobs, (2) suspend one or more jobs, (3) kill one or more jobs, or (4) no action at all. Any combination of the first three or the fourth is possible depending on the procedure. The selected actions are sent to the Server.

When the scheduling cycle is complete, the Scheduler closes the connection to the Server and waits for a new connection, which starts a new scheduling cycle.

3.2.4: Scheduler Implementations

PBS provides two different sets of tools which allow the site to implement its scheduling policies in two different methods. The two methods trade off capabilities and complexity.

The BASL Scheduler -

The first method is a complete yacc - lex based scheduler program that is an interpreter of a PBS defined language. The site writes a scheduling script in the BAtch Scheduling Language (BASL). BASL is a procedural language that is similar in syntax to C. The main features of BASL are:

- offers variables to declare and hold the data required from the Server, from the Resource Monitor, time and date information, and local data.
- supports string concatenation and arithmetic operations.

- provides for decision making and looping over the range of hosts, queues, and jobs through control constructs.
- provides ability to command the Server through action statements.

When the Scheduler starts, it reads and parses the provided script. Then, on each scheduling cycle, the Scheduler gathers the information required and executes the script, evaluating the data and directing the Server to take action.

Tcl Scheduler -

The other supplied method is a set of Tool Control Language (Tcl) routines that allow the site to write a Tcl based scheduler.[6] The routines are provided which supply the scheduler framework and interface to both the Scheduler and the Resource Monitor.

The implementation of a scheduler in Tcl is more complex than in BASL, but the site gains flexibility that allows the implementation of a wider range of policies. An example of this flexibility is given when discussing the current SP-2 scheduling later in this article.

Yet another alternative -

If neither of the two supplied implementation methods is satisfactory, a site may use the supplied APIs and implement a complete scheduler program in the C, or similar language. This ability to replace the scheduling controls without having to modify the basic batch system provides both a systems administrator site-specific controls and a researcher with a excellent tool to test scheduling theories.

4: Scheduling Examples

The best demonstration of the power of scheduling in PBS is some examples of both BASL and Tcl. These examples show the power of scheduling when full information about the jobs and system resources is available and the policy implementation is not bound by limited controls.

4.1: Serial Jobs

The following is a very simple BASL script to introduce the language. It is not very practical, but is a place to start. The script will run up to 3 jobs if the load average is less than 2.0:

```
global variable nrun;                                    1
host resource loadave;                                   2
job requirement cput jcput;                              3
                                                         4
rm this_host;                                            5
                                                         6
foreach host {                                           7
 nrun = 0;                                               8
 if (loadave < 2.0) {                                    9
  foreach job {                                          10
   if (jcput < 60) {                                     11
    run;                                                 12
```

```
    nrun += 1;                                                    13
   }                                                              14
   if (nrun > 2) exit;                                            15
  }                                                               16
 }                                                                17
}                                                                 18
```

Lines 1 through 3 declare variables. In line 1 "`global`" indicates that the variable is defined and initialized within the script and "`variable`" indicates that the variable is a simple type. On line 2, "`host`" indicates the data for this variable is obtained from the resource monitor, "`resource`" is the type, and "`loadave`" is the name of the resource and the variable name. Line 3 declares the variable with an alias of "`jcput`" whose source is the job status query to the server, whose type is "`requirement`" (one of the listed job resource requirements), and whose requirement name is "cput" (CPU time). Both host and job variables are actually arrays with an entry for each host and job respectively. Line 5 specifies on which host (one named "this_host") the Resource Monitor is found.

Line 7 starts the procedure by indexing the "host" variable through each member of the array. Thus line 9 examines the loadave on each host. Line 10 loops once for each job for which status was returned from the server. Thus in line 11 the "CPU time" requirement of each job is checked in turn. If it is less than 60 seconds, line 12 directs that the job should be run.

Line 15 stops the process if 3 jobs have been scheduled during this cycle.

4.2: Serial Jobs on a Cluster

In the prior section a BASL script was explained that scheduled jobs based on load average on a single host. Here that script is expanded to schedule serial jobs on a cluster of five systems:

```
global variable nrun;                                             1
host resource loadave;                                            2
job requirement cput jcput;                                       3
rm host1;                                                         4
rm host2;                                                         5
rm host3;                                                         6
rm host4;                                                         7
rm host5;                                                         8
                                                                  9
foreach host {                                                   10
 nrun = 0;                                                       11
 if (loadave < 2.0) {                                            12
  foreach job {                                                  13
   if (jcput < 60) {                                             14
    run JID HID;                                                 15
    nrun += 1;                                                   16
   }                                                             17
   if (nrun > 2) exit;                                           18
```

```
  }                                                              19
 }                                                               20
}                                                                21
```

The only change required was to add the various host names, host1 through host5, and to modify the run statement, line 16, to specify the current job and current host. Of course, the script ignores several factors that would be important in a real cluster, such as workstation architecture, time of day, and use by the workstation owner. A few simple additions will take care of those factors as shown below:

```
global variable nrun;                                            1
host resource loadave;                                           2
host resource arch harch;                                        3
host resource idletime;                                          4
job requirement cput jcput;                                      5
job requirement arch jarch;                                      6
                                                                 7
rm host1;                                                        8
rm host2;                                                        9
rm host3;                                                       10
rm host4;                                                       11
rm host5;                                                       12
                                                                13
foreach host {                                                  14
 nrun = 0;                                                      15
 if (loadave < 2.0) {                                           16
  foreach job {                                                 17
   if(((jarch=="any")||(jarch==harch))                          18
     &&                                                         19
     (idletime > 15m) ) {                                       20
    if ((DAY>=MON) && (DAY <= FRI) &&                           21
        (NOW > 7:00:00) &&                                      22
        (NOW < 18:00:00)) {                                     23
     if (jcput < 60) {                                          24
      run JID HID;                                              25
      nrun += 1;                                                26
     }                                                          27
    } else if (jcput < 8h) {                                    28
     run JID HID;                                               29
     nrun += 1;                                                 30
    }                                                           31
   }                                                            32
   if (nrun > 2) exit;                                          33
  }                                                             34
 }                                                              35
}                                                               36
```

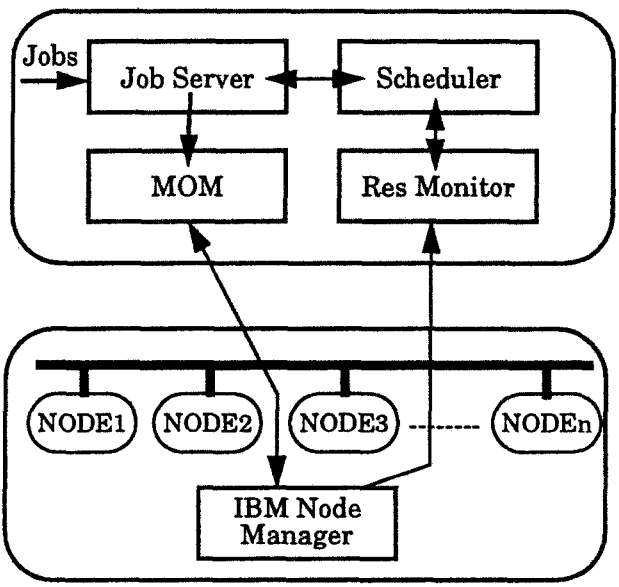
4.3: Parallel Jobs on a Parallel System

PBS supports several parallel systems including the CM-5 and Paragon. The newest is NAS's IBM SP-2 which is composed of 160 nodes. 144 nodes are available to the users and 16 are reserved for system development. Basically, the SP-2 is a workstation cluster with a high speed switch interconnect.

The IBM SP-2 has two modules of system software that provides the image of a single system. The IBM Job Manager allocates and frees nodes for all jobs in the system. The IBM Parallel Operating Environment software, *poe*, is designed to be used by the interactive user. The user indicates the number of nodes required or provides a list of node names. Poe attempts to obtain the required nodes from the Job Manager, and sets up the message routing switch.

Figure 3 shows that PBS considers the SP-2 to be a single system. There is only one instance of MOM and of the Resource Monitor. In fact, PBS is not even on the SP-2, but runs on a separate control workstation. Both MOM and the Resource Monitor interact with the IBM Job Manager through the provided API.

Figure 3. PBS on the IBM SP-2



To add to the complexity of the system, not all nodes on the SP-2 are identical. Some have more memory or disk, and some have a HIPPI connection. The IBM Job Manager is not aware of any differences, but PBS is. A "node list" file provided by the administrator has an entry for each node. The minimum information per entry is the name of the node. Nodes that are distinguished by some property also have that property listed in their entry.

A PBS user specifies the node requirement of the job when the job is submitted. The specification can be as simple as a number of nodes required or as complex as a

multiple set of properties required. For example, to ask for 16 nodes, the user would submit a job as follows:

```
qsub -l nodes=16
```

If a user needs two nodes with a HIPPI connection, one node with extra memory, one node with extra memory and disk space, and 12 "plain" nodes, totalling 16, the request would be

```
qsub -l nodes=2:hippi+mem+mem:disk+12
```

When this job is considered for scheduling, the scheduler sends the node requirement string for each job to the Resource Monitor asking if the nodes are available. The Resource Monitor reads the node file and maps the requested properties to node names. The Resource Monitor then obtains the status of all nodes from the Job Manger and performs a best fit. For each node request, the Resource Monitor returns one of three responses, yes, no, or never. If the nodes for a given job are available and if the job meets the other scheduling criteria, the Scheduler directs the Server to run the job. MOM then makes the node list available to the job. Within the job script, the user invokes a wrapper which calls poe with the host list and the nodes are assigned to the job.

4.3.1: BASL Script for SP-2

Below is a sample BASL script to implement a scheduling policy used at NAS. It is presented to demonstrate that PBS easily handles complex policies. The policy is:

1. Prime time, 6 AM to 6 PM
   a. When less than 113 nodes in use:
      1-32 node jobs limited to < 4 hours.
      >32 node jobs limited to < 10 minutes.
   b. When more than 112 nodes are already in use:
      jobs limited to < 10 minutes. This maintains high availability on the last 32 nodes.
2. Interactive Extension Period, 4 AM to 6 AM and 6 PM to 10 PM
   a. When less than 113 nodes in use:
      1 - 128 node jobs limited to < 6 hr.
      > 128 node limited to < 10 min.
   b. Jobs using last 16 nodes are limited to less than 10 minutes
   c. Jobs are not started if they might not complete before the end of the shift.
3. Night time, 10 PM to 4 AM Monday through Friday and all day Saturday and Sunday.
   a. 1-144 node jobs limited to < 6 hours.
   b. Jobs are not started if they might not complete before the end of the shift.

Note that all nodes are used exclusively by one job at a time, i.e. space sharing. This policy reduces message passing latency between nodes assigned to the job.

```
rm 15003 sp2                                                    1
job attribute job_state;                                        2
job attribute queue_type;                                       3
job requirement nodes;                                          4
job requirement nodect;                                         5
job requirement walltime;                                       6
job resource avail:~ID:nodes anodes;                            7
host resource totpool;                                          8
host resource usepool;                                          9
                                                                10
global variable prime;                                          11
                                                                12
foreach host {                                                  13
 if((DAY >= Mon) && (DAY <= Fri) &&                             14
    (NOW >=6:00:00) &&                                          15
    (NOW < 18:00:00)) {                                         16
                                                                17
# Prime Time                                                    18
                                                                19
  foreach job {                                                 20
   if (job_state == "Q") {                                      21
    if ((totpool - usepool) > 32) {                             22
     if ((nodect<33)&&(walltime>4h))                            23
      continue;                                                 24
     if ((nodect>32)&&(walltime>10m))                           25
      continue;                                                 26
    } else {                                                    27
     if ((nodect>=32)||(walltime>10m))                          28
      continue;                                                 29
    }                                                           30
    if (anodes =="yes") {                                       31
     run;                                                       32
     break;                                                     33
    } else if (anodes == "never")                               34
     delete JID REASON;                                         35
   }                                                            36
  }                                                             37
 } else if ((DAY>=Mon)&&(DAY<=Fri) &&                           38
    ((NOW>=4:00:00)&&(NOW<16:00:00)) ||                         39
    ((NOW>=18:00:00)&&(NOW<22:00:00))){                         40
# Interactive night                                             41
  foreach job {                                                 42
   if ((job_state == "Q") &&                                    43
       (queue_type == "E")) {                                   44
```

```
    if ((totpool-usepool)>16) {                                    45
     if ((nodect<129)&&(walltime>6h))                              46
      continue;                                                    47
     if ((nodect>128)&&(walltime>10m))                             48
      continue;                                                    49
    } else {                                                       50
     if ((nodect>=16)||(walltime>10m))                             51
      continue;                                                    52
    }                                                              53
    if (NOW < 06:00:00) {                                          54
     if ((DAY!=Sat) && (DAY != Sun) &&                             55
        (walltime >(06:10:00-NOW)))                                56
        continue;                                                  57
    }                                                              58
    if (anodes == "yes") {                                         59
     run;                                                          60
     break;                                                        61
    } else if (anodes == "never")                                  62
     delete JID REASON;                                            63
   }                                                               64
  }                                                                65
 } else {                                                          66
# Batch Night                                                      67
  foreach job {                                                    68
   if ((job_state=="Q") &&                                         69
       (queue_type == "E")) {                                      70
    if ((nodect>144)||(walltime> 6h))                              71
     continue;                                                     72
    if (NOW < 06:00:00) {                                          73
     if ((DAY!="Sat")&&(DAY!="Sun")&&                              74
       (walltime > (06:10:00-NOW)))                                75
       continue;                                                   76
    }                                                              77
    if (anodes == "yes") {                                         78
     run;                                                          79
     break;                                                        80
    } else if (anodes == "never")                                  81
     delete JID REASON;                                            82
   }                                                               83
  }                                                                84
 }                                                                 85
}                                                                  86
```

The statement "`delete JID REASON;`" as found on lines 35, 63, and 82 directs the Server to delete the job and pass the reason given in the special variable "`REASON`" to the user. Note on line 75, that jobs are allowed to extend 10 minutes beyond the

night time period. This is done because the policy states that large jobs can run up to 10 minutes during this time.

4.3.2: Tcl Script for the SP-2

The current job scheduling policy for the NAS SP-2 is similar to the one given in section 4.3.1 above. It is implemented using the Tcl based scheduler. The new policy adds one major capability, automatic inclusion of scheduled down time.

A program already exists which is used to communicate scheduled dedicated and maintenance time to the general user community. The data file for this program is updated whenever the schedule changes. With the Tcl script, it is not necessary to duplicate the updates into the scheduling program, the script automatically invokes the existing program and processes the output. When considering which jobs should be started, the Tcl script will ignore any which would run into the next scheduled down time.

This is an example of a policy implementation in PBS that includes an ability that was unplanned in the beginning, yet required no changes to the underlying batch system. Scheduling policy development at the NAS Facility have been pragmatic. The goal is to insure throughput rather than perform research into scheduling algorithms. PBS has proven itself extremely useful in that environment.

5: Scheduler Performance

A concern was raised about the performance of the PBS scheduler. Does the separation of the scheduler or the use of a script language cause the scheduler to take too much time?

Testing on several systems at the NAS Facility have not shown this to be a problem. In fact, the responsiveness to the arrival of new jobs seems to be superior to the vendor batch system. On the SP-2 under a production load of up to 50 jobs queued, the Tcl based scheduling cycle has not taken more than 5 seconds. The typical time is 1 to 3 seconds. This is a small amount of overhead for jobs that run for many minutes or hours.

Timing on an IBM 590 workstation using BASL shows the correlation between cycle time, the number of jobs, and the complexity of the script. Table 1 summaries the averages from a number of timing tests. Two scripts, one trivial and one fairly complex, were measured against a queue with a length of 1, 10, 100, and 1000 jobs.The trivial case timing establishes the overhead of obtaining the job information from the Job Server. The timings for the non-trivial case show the impact of adding complexity to the policy.

Timings for BASL on a Cray C90 show similar numbers. Due to production constraints, only the timings for the trivial case and only up to 100 jobs were gathered as show in Table 2. These timings where taken on a loaded Cray whereas the 590 timings where on a lightly loaded system.

Table 1: Scheduler Cycle Time on IBM 590
(in seconds)

Jobs	Trivial	Non-trivial
1	0.02	0.04
10	0.28	0.35
100	1.0	1.8
1000	12.	15.

Table 2: Scheduler Cycle Time on IBM 590
(in seconds)

Jobs	Trivial	Non-trivial
1	0.02	0.04
10	0.28	0.35
100	1.0	1.8
1000	12.	15.

Based on these numbers for BASL and the experiences on the SP-2 with Tcl, scheduler performance does not appear to be a problem in PBS.

6: Future Work

Two efforts are under way to extend the support PBS offers for parallel jobs. First is to provide support for "interactive" jobs scheduled through PBS. An "interactive" batch job is one that is submitted to PBS and scheduled but the standard streams of the job are connected to the submitter's terminal when it is run. This allows the user to run interactive tools such as debuggers while the batch system is providing scheduling and resource management for all jobs.

The second major effort is to extend PBS to fully support parallel jobs on clusters. This will allow for the scheduling and reservation of multiple hosts per job and the creation of parallel tasks on those nodes. PBS will provide for shared use of nodes (in which jobs are time-shared on the assigned nodes) and for exclusive use of nodes. Extensions to BASL are being designed to enable the Scheduler to interpret the node requirement expression, determine node availability from multiple Resource Monitors, and to generate a node list which satisfies the requirement.

7: Acknowledgments

PBS was developed as a joint project between the *NAS Systems Division* at NASA Ames Research Center and the *National Energy Supercomputer Center* and *Livermore Computer Center* at the Lawrence Livermore National Laboratories. The following past and current members of the PBS development team deserve special recognition:

- NASA Ames Research Center employees:
  Dave Tweten and John Musch for their leadership and support.
- Computer Sciences Corporation employees at Ames Research Center:
  Tom Proett developer of most of the machine dependent code in the Resource Monitor and MOM and the Tcl scheduler.
- Lawrence Livermore National Labs employees:
  Kent Crispin and Terry Heidelberg, libraries and commands contributors; Bruce Kelly, commands and the C90 port implementor; special acknowledgment to Clark Streeter, implementor of the Scheduler BASL language

8: References

[1]Kingsbury, B. *"The Network Queuing System"*, Sterling Software, Palo Alto.

[2]Mutka, M. *"Sharing in a Privately Owned Workstation Environment."* Ph.D. thesis, University of Wisconsin, May 1988.

[3]Litzkow, M., Livny. M. and Mutka, M. *"Condor — A Hunter of Idle Workstation"*. Proceeding of the 8th International Conference on Distributed Computing Systems. San Jose, CA. June 1988.

[4]Duke, D., Green, T., and Pasko, J. *"Research Toward a Heterogeneous Networked Computing Cluster: The Distributed Queuing System Version 3.0"*. Supercomputing Computations Research Institute, Florida State University, March 2, 1994.

[5]Zhou, S., Zheng, X., Wang, J., and Delisle P. *"Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems"* Software - Practice and Experience, Volume 23, December 1993.

[6]Ousterhout, J., *"Tcl and the Tk Toolkit". Addison-Wesley Publishing. 1994.*