

ARGONNE NATIONAL LABORATORY  
9700 South Cass Avenue  
Argonne, IL 60439

---

ANL/MCS-TM-201

---

## **Users Guide to the Argonne SP Scheduling System**

by

*David A. Lifka,\* Mark W. Henderson, and Karen Rayl*

Mathematics and Computer Science Division

Technical Memorandum No. 201

May 1995

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

\* Also affiliated with the Illinois Institute of Technology.

# Contents

<b>Abstract</b>	<b>iv</b>
<b>1 SP Scheduler Policy Overview</b>	<b>1</b>
<b>2 SP Scheduler Users Guide</b>	<b>1</b>
2.1 Terminology . . . . .	1
2.2 Scheduler Commands . . . . .	1
2.3 Getting Started . . . . .	6
2.4 Supported Job Types . . . . .	9
2.5 Using Temporary Storage on the SP Nodes with the Scheduler . . . . .	12
2.6 Releasing Resources When Finished . . . . .	12
2.7 Current Queuing Algorithm . . . . .	12
<b>3 System Installation and Administration Guide</b>	<b>15</b>
3.1 Installing the Scheduler . . . . .	15
3.2 Modifying the Scheduler for a Specific Environment . . . . .	28
3.3 System Administration Tools . . . . .	29
<b>4 Scheduler Architecture</b>	<b>30</b>
4.1 Components of a Scheduling System . . . . .	30
4.2 SP Scheduler Design . . . . .	31
4.3 Sample Flow of Execution . . . . .	33
<b>References</b>	<b>33</b>

# Users Guide to the Argonne SP Scheduling System

by

*David A. Lifka, Mark W. Henderson, and Karen Rayl*

## Abstract

During the past five years scientists discovered that modern UNIX workstations connected with ethernet and fiber networks could provide enough computational performance to compete with the supercomputers of the day. As this concept became increasingly popular, the need for distributed queuing and scheduling systems became apparent. Today, supercomputers, such as Argonne National Laboratory's IBM SP system, can provide more CPU and networking speed than can be obtained from these networks of workstations. These modern supercomputers look like clusters of workstations, however, so developers felt that the scheduling systems that were previously used on clusters of workstations should still apply. After trying to apply some of these scheduling systems to Argonne's SP environment, it became obvious that these two computer environments have very different scheduling needs. Recognizing this need and realizing that no one has addressed it, we developed a new scheduling system. The approach taken in creating this system was unique in that user input and interaction were encouraged throughout the development process. Thus, a scheduler was built that actually worked the way the users wanted it to work. This document serves a dual purpose. It is both a user's guide and an administrator's guide for the ANL SP scheduling system. Look for revisions to this guide that will be appearing.

# 1 SP Scheduler Policy Overview

The goals of the Mathematics and Computer Science Division's Argonne SP job scheduler are fairness, simplicity, and efficient use of the available SP resources. These goals are in conflict, but the scheduler is designed to be a compromise. Users will be able to request a set of nodes for any type of use. In order to maintain the quality of machine access, the scheduler provides a single point of access, `spsubmit`. This program will allow users to queue both interactive and batch access to the SP. When resources are available, the user will be notified by the scheduler and at that time will have *exclusive* access to the number of nodes requested. Having *exclusive* access to the SP nodes allows the user to have optimum cache performance and use of all available memory and `/tmp` disk space. This type of access allows users to run benchmarks at any time and also to predict how long it will take for their job to complete. Having exclusive access is essential so that users can predict wall-clock run time for their jobs when they submit them to the scheduler. While there are currently no limits to the number or size of jobs that can be submitted, the scheduler uses a public algorithm to determine when batch or interactive time is actually provided (see Section 2.9). Any modifications to this algorithm will be made public. To view the current queuing policy, use the command `spq -l`. MCS has also implemented an allocation policy as a separate part of the scheduler. The intent of the policy is to ensure all users some set amount of resource time and to prevent people from using more than their share of resources.

## 2 SP Scheduler Users Guide

### 2.1 Terminology

A few terms and conventions are used throughout this document. Commands that can be typed at the Unix command line will be in a typewriter-like font: `command_name`. "CAC" stands for charge allocation category and is the account unit for the scheduler accounting system developed at Argonne for use with the scheduler. It currently is used only at Argonne and has not been released for distribution. If you are using this users guide somewhere other than Argonne you can ignore any reference to the accounting system or CAC. GUI stands for graphical user interface and is typically used in reference to `xspusage`.

### 2.2 Scheduler Commands

The scheduler was designed to be easy to use and to understand. The user interface to the scheduler is made up of a group of commands that behave much like existing Unix commands. This section lists the various user commands for using the SP job scheduler and gives a brief explanation of their function. *NOTE: All of the commands have "-h" options which provide help and describe additional functionalities they may have.*

#### 2.2.1 `spq`

`spq`: Displays the SP scheduler job queue. It provides the following job information:

- the job IDs for each job
- the username associated with each job
- the number of nodes required/used by each job
- which jobs will start during the day, night, or weekend
- whether the job is (I)nteractive or (B)atch
- the current status of each job
  - (W)aiting to run

- (P) paused by the user
- (H) held by the system
- (h) held by the system and paused by the user
- how much time the job is requesting or when it will finish if it is currently running.

Here is an example of `spq` in use:

```

bonnie.mcs.anl.gov% spq
*****
Please report any problems to spsupport@mcs.anl.gov
*****

115 nodes Available          0 nodes Down

-----
Job ID      User      # Of      Job      Req./Stop
Number      Name      Nodes  Queue  Type  Status  Date  Time
-----
06154111    panigrah      3  Running  N/A    R      12/06 17:41
06173226    kohr          2  Running  N/A    R      12/06 17:52
06172525    michalak      4  Running  N/A    R      12/06 17:55
06172652    tuecke        4  Running  N/A    R      12/06 17:57
05162926    moon         128   Day      B      W      0day  0:05
05162940    moon         128   Day      B      W      0day  0:05
06160605    nakano        16   Day      B      W      0day  2:00
06165839    bryant        36   Day      B      W      0day  0:45
06170615    asussman      30   Day      B      W      0day  0:45
06173718    wiringa      64   Day      B      W      0day  0:20
06173728    spieper      32   Day      B      W      0day  0:20
06173828    spieper      39   Day      B      W      0day  0:15
06173816    twang         9    Day      I      W      0day  1:00
06064052    chasman      32   Night    B      W      0day  6:40
06064909    chasman      32   Night    B      W      0day  4:00
06070726    chasman      32   Night    B      W      0day  4:00
06094447    nanjundi     64   Night    B      W      0day  3:00
06123845    leaf         16   Night    B      W      0day  6:00
06172912    pudliner     32   Night    B      W      0day  1:40
06150129    chasman      32   Night    B      H      0day  4:00
bonnie.mcs.anl.gov%

```

### 2.2.2 spsubmit

**spsubmit:** Used to submit jobs to the queue. This command will prompt you for information about your job. Once you have answered all the questions, it will ask you to verify your answers and submit that job to the queue. Once the job has been submitted, this command will return your unique job ID for that job, which you can use to track your job using `spq`. Any information the system returns to you about this job will also contain this job ID.

### 2.2.3 srelease

**srelease:** Is used to remove jobs from either the queue or the machine if the job has already started. If your job is currently running, it informs the scheduler that your node(s) can be returned to the free pool and that you should no longer be charged against your charge allocation category (CAC). Once you **srelease** the nodes you are on, it may take up to three minutes to release them

all, depending on how many you are releasing. This is because the scheduler, before returning nodes to the free pool, verifies that you have left them in a usable state. You are *not* charged for the release time. After **sprelease** finishes issuing all the release requests to the scheduler, it returns the number of nodes that it has successfully released.

**sprelease** can be used in two ways. The first is when it is issued with a specific job ID, for example,

```
% sprelease 09144216
```

In this case all nodes in use by this particular job will be released to the free pool. If issued with a job ID, **sprelease** can be issued from any workstation that has the scheduler running on it including the nodes. The second way **sprelease** can be used is without a job ID when run from a particular node that you have been allocated. This is useful for nonparallel jobs that may finish with particular nodes before others and wish to return them to the free pool while the others continue to be used.

#### 2.2.4 spfree

**spfree:** Is most commonly used to return the number of nodes currently available for use. It has the following options:

- h Lists all the options and their functions.
- f <Default> Prints the number of currently available nodes.
- d Prints the number of down nodes.
- r Prints the number of nodes currently reserved for demonstrations.
- g Prints the number of nodes available without going through the scheduler.
- r Prints the number of nodes that are currently in use.

#### 2.2.5 sppause

**sppause:** Changes the status of all the user's interactive jobs in the queue from (W)aiting to (P)aused causing them to be passed over by the scheduler for execution until the user will be able to use the interactive computer time at some point in the future. If a job has a status of "h" after being paused, that job has been paused and also has been (H)eld by the system because the user has requested more resources than the scheduling policy allows to be scheduled at any particular time.

#### 2.2.6 spunpause

**spunpause:** Changes the status of all the user's interactive jobs in the queue from (P)aused to (W)aiting so that the scheduler will start the job as soon as the resources the job needs are available. If a job with the status of "h" is unpaused, it will change to "H" until the number of outstanding resources requested is reduced either by some of the jobs running or by some of the queued requests being removed.

#### 2.2.7 spstatus

**spstatus:** This command is updated by the system administrator(s) and contains the current status of the SP. Here is some sample output from **spstatus** on the Argonne SP.

```
bonnie.mcs.anl.gov% spstatus
SP Status:
-----
12/13/94
- 126 nodes are up and running
```

- The SP system has been upgraded to SP software version 2.1. The new POE libraries have been loaded onto bonnie and clyde and users must recompile their code in order for it to work properly.

SP Downtime:  
-----

- Wednesday 12/14 Racks 1 and 5 will be out of service while we finish the installation of TB2 adaptors in Mercury (node 1) and Jupiter (node 65). Cables we received were bad causing us to have to wait 1 more day for these nodes to be installed. After the installation bonnie and clyde will be rebooted as well. Kingston will be looking into the memory allocation problem, the slow compile times on bonnie and clyde, and the p4 initialization problems throughout the week. Nodes which show in "service" mode are being used to debug these problems.
- Next planned downtime is on 12/20 from 8:00 am to 12:00 pm for regular system maintenance  
bonnie.mcs.anl.gov%

### 2.2.8 spusage

**spusage:** Shows who is currently using which nodes. The format of its output format is node\_number(1,0,-1,-2,-3 -4,-5);username;Job ID;Job run-time. The second number signifies the status of the node.

- 1 The node is being used by a user job.
- 0 The node is available.
- 1 The node is down and thus unavailable.
- 2 The node is being serviced and thus unavailable.
- 3 The node is generally available for all users.
- 4 The node is being used for a CAVE demonstration.
- 5 The node is being used for a demonstration.

### 2.2.9 xspusage

**xspusage:** Displays who is currently using which nodes. An X Windows GUI interface, **xspusage**, performs the same function as the ASCII **spusage** command. It also provides a way for a user to quickly determine which nodes they have been allocated. The user can then click on those nodes in this X interface to open xterms on those nodes. This program was based on the GUI tools that were developed as part of the *Scalable Unix Tools* project [2]. The MPI Implementation, MPICH, has tools like this as well as tools to automate the submission and execution of jobs using the SP scheduler [4].

Here is a snapshot of **xspusage**:

xspusage							
Small				Quit			
Service	wiringa	open	open	pudliner	open	panigrah	pudliner
dstevens	wiringa	wiringa	open	pudliner	open	panigrah	pudliner
wiringa	wiringa	wiringa	open	pudliner	open	krishnai	open
wiringa	dstevens	wiringa	open	pudliner	open	krishnai	open
wiringa	dstevens	spieper	open	pudliner	open	krishnai	wiringa
dstevens	wiringa	open	open	pudliner	open	krishnai	panigrah
wiringa	dstevens	open	open	pudliner	pudliner	rbennett	open
wiringa	wiringa	open	Service	krishnai	pudliner	rbennett	open
wiringa	wiringa	open	spieper	pudliner	pudliner	rbennett	open
wiringa	wiringa	open	open	pudliner	pudliner	rbennett	open
wiringa	wiringa	open	open	pudliner	krishnai	rbennett	open
wiringa	wiringa	open	wiringa	open	krishnai	rbennett	open
wiringa	wiringa	open	wiringa	open	krishnai	rbennett	open
wiringa	wiringa	open	wiringa	open	krishnai	rbennett	Demo
wiringa	leaf	open	wiringa	open	krishnai	rbennett	General
wiringa	homann	open	wiringa	open	panigrah	rbennett	Cave
49							
Mon 14:31							

Following is a snapshot of **xspusage** in small mode. This mode is useful if you just want to watch for a large group of nodes to become available. You can leave this application in the corner of your screen and easily monitor the system by watching the color changes.

xspusag	
Large	
Quit	
33	
Mon 14:33	

The colors correspond to the the numeric values in **spusage** and indicate node status.

- |    |   |            |
|----|---|------------|
| 1  | The node is use by a user job                   | blue       |
| 0  | The node is available                           | yellow     |
| -1 | The node is down and thus unavailable           | red        |
| -2 | The node is being serviced and thus unavailable | light blue |
| -3 | The node is generally available for all users   | green      |
| -4 | The node is being used for a CAVE demonstration | brown      |
| -5 | The node is being used for a demonstration      | pink       |



### 2.2.10 spwhat

**spwhat:** Shows the number of nodes currently available and how long they will be available. If a job is submitted based on the information provided by **spwhat**, it should start immediately. Currently, **spwhat** has not been updated to work with the latest scheduling algorithm and so it is turned off. It should be fixed and available soon.

### 2.2.11 spwhen

**spwhen:** Estimates when a particular job, referred to by Job ID, will start based on the current queuing algorithm. This is a worst-case estimate in that the job will start no later than this time but *may* be started sooner if jobs ahead of it in the queue finish before they are expected to. Currently, **spwhen** has not been updated to work with the latest scheduling algorithm and so it is turned off. It should be fixed and available soon.

### 2.2.12 spwait

**spwait:** Provides a way to "watch" a particular Job ID in the queue and to return when the job is no longer in the queue. This is useful for users whose jobs require the output from a previous job as input to the next. Scripts can be written to verify the output of the job, once a job has completed, and then to automatically submit the next job to the queue with the appropriate input. If a user has one job in the queue, **spwait** will "watch" that job. If it is necessary to monitor several jobs, **spwait** can be executed with a Job ID and each invocation will return when the job it is "watching" is no longer in the queue.

### 2.2.13 getjid

**getjid:** Returns the Job ID based on the node it is executed on. This is a useful command for batch scripts that need to determine which nodes they are running on after they've been started on them.

### 2.2.14 cacReport

**cacReport:** Provides statistics for a given CAC.

### 2.2.15 what\_cac

**what\_cac:** Reports the CAC group of a given user ID.

## 2.3 Getting Started

The **spsubmit** command will allow you to submit jobs to the scheduling system. The current scheduler will prompt you for necessary information about your job. *NOTE: In the future there will be a version of **spsubmit** that supports command-line arguments.* The scheduler will assign your job entry a unique numeric job ID. This unique job ID can be used to track how many jobs you have submitted to the scheduler and also to remove your unwanted job entries from the scheduler by using **spq** and **sprelease**, respectively. When the nodes you've become available, the scheduler will create a special file in your **/sphome** directory. The scheduler will also notify you via e-mail at the time the nodes you requested were actually allocated and the names of the nodes that were assigned to you. You can also use **xspusage** to determine which users are using which nodes. The file has the form:

```
/<home_path>/<username>/SPnodes.<job_ID>
```

Here is an example:

`/sphome/lifka/SPnodes.09145115`

This file contains a list of the nodes names you have been assigned for a particular run. Nodes allocated to you by the scheduler will be in exclusive access mode. Only your login will be enabled on the nodes that you are allocated. (This means you will be the only one that can log into these nodes until you release them with **sprelease**. Once you issue the **sprelease** on the nodes allocated to you (see Section 2.8), you will not be able to log into those nodes.)

### 2.3.1 Batch Submission with **spsubmit**

If you have submitted a program or script for batch execution, it will be automatically executed for you. It does this by issuing an **rsh** of the program or script to the first node in the group that it allocates to you.

If you require that data be staged to nodes or other job setup, you should submit a job script that performs the appropriate setup and then runs your program. Be sure when submitting programs or scripts that you provide the full path to them, for example,

```
(/sphome/<user_name>/<program>)
```

You should also use full paths to any files or programs referenced inside your programs or job scripts.

*NOTE: even though a job is running as a batch job, the user who submitted it will have access to the nodes it is executing on.*

### 2.3.2 Interactive Submission with **spsubmit**

It may be useful to request interactive time on the SP until you have thoroughly tested your programs and/or scripts. This will allow you to closely monitor their performance until you are confident that they will run correctly in batch mode.

If you schedule interactive time and it gets queued because the system is currently loaded, be sure that you will be available to use the resources when they actually become available. For example, if the system is heavily loaded and the interactive time you requested becomes available at midnight, you will need to be able to use those resources at midnight because they will be allocated to you and thus billed against your CAC until they are released with **sprelease**. You may wish to remove interactive jobs or pause them with **sppause** from the queues when leaving work for the day.

When you use **spsubmit** to request interactive time, it asks for information about your job even though you can use the nodes any way you wish interactively. The scheduler is just a means of collecting this information to study how people are using the machine.

### 2.3.3 **Spsubmit** Example 1: An MPL Script

We present a submission example in which the user wishes to submit a script that runs an MPL program in "user space" mode over the IBM switch.

```
bonnie.mcs.anl.gov% spsubmit
Charge Allocation Category: [default lifka]
Maximum Wall-clock Run-Time (minutes): (1-???) 15
Number of Nodes Required: (1 - 128) 4
CAC: "lifka" *now* has 28114 RUs available, after committing.
(I)nteractive (B)atch: b
Job Classification (M)PL, (T)ask Farm, (P)4: m
IP over the switch [y/n]: n
Full path to Shell Script/Program: /sphome/lifka/MPL/RunMe
Command Line arguments for your job:
```

```
Username: lifka
Charge Allocation Group: lifka
Job Type: B
Job Classification: mpl
IP over the switch: no
Number of Nodes: 4
Maximum Wall-clock Run-Time (minutes): 15
Program/Job Script: /sphome/lifka/MPL/RunMe
Command Line Arguments:
```

```
(C)ommit or (A)bort: c
```

```
Your JID for this job is: 12132612
bonnie.mcs.anl.gov%
```

### 2.3.4 Ssubmit Example 2: A p4 Program

Our second submission example is for a p4 program, not a script. Notice that the p4 debugging options and output redirection were entered as desired command-line arguments.

```
bonnie.mcs.anl.gov% ssubmit
Charge Allocation Category: [default lifka]
Maximum Wall-clock Run-Time (minutes): (1-???) 15
Number of Nodes Required: (1 - 128) 16
CAC: "lifka" *now* has 27814 RUs available, after committing.
(I)nteractive (B)atch: b
Job Classification (M)PL, (T)ask Farm, (P)4: p
IP over the switch [y/n]: y
Full path to Shell Script/Program: /sphome/lifka/P4/UniTree/Utest
Command Line arguments for your job:-p4dbg 10 -p4rdbg 10 > /sphome/lifka/run.out
```

```
Username: lifka
Charge Allocation Group: lifka
Job Type: B
Job Classification: p4
IP over the switch: yes
Number of Nodes: 16
Maximum Wall-clock Run-Time (minutes): 15
Program/Job Script: /sphome/lifka/P4/UniTree/Utest
Command Line Arguments: -p4dbg 10 -p4rdbg 10 > /sphome/lifka/run.out
```

```
(C)ommit or (A)bort: c
```

```
Your JID for this job is: 12135506
bonnie.mcs.anl.gov%
```

### 2.3.5 Ssubmit Example 3: A Task Farm Script

```
bonnie.mcs.anl.gov% ssubmit
Charge Allocation Category: [default lifka]
Maximum Wall-clock Run-Time (minutes): (1-???) 15
Number of Nodes Required: (1 - 128) 32
CAC: "lifka" *now* has 26526 RUs available, after committing.
(I)nteractive (B)atch: b
Job Classification (M)PL, (T)ask Farm, (P)4: t
```

```

IP over the switch [y/n]: n
Full path to Shell Script/Program: /sphome/lifka/RunMe
Command Line arguments for your job:4 16

```

```

Username: lifka
Charge Allocation Group: lifka
Job Type: B
Job Classification: task
IP over the switch: no
Number of Nodes: 32
Maximum Wall-clock Run-Time (minutes): 15
Program/Job Script: /sphome/lifka/RunMe
Command Line Arguments: 4 16

```

```

(C)ommit or (A)bort: c
Your JID for this job is: 12135557
bonnie.mcs.anl.gov%

```

## 2.4 Supported Job Types

With the SP scheduler, nodes can be used both for interactive and batch computing. The nodes are configured to run parallel programs that use the IBM switch or ethernet for communication. Jobs that use MPL, MPI, or p4 for interprocess communication fall into this category. Task farm jobs, or those that use the SP as if it were a large cluster of network-connected IBM RS6000s, are also supported. *If you are not sure how to classify a program, please consult the ANL SP Users Guide [5].* If you are still having trouble, contact [spsupport@mcs.anl.gov](mailto:spsupport@mcs.anl.gov) or your local support staff.

### 2.4.1 MPL

MPL is the new IBM switch software that replaces EUI/EUIH. To use MPL, it is preferable to use mpcc and mpplx instead of xlc and xlf to compile and link your programs. For more details on compiling and linking MPL code, see the *SP Users Guide*. To run an MPL code, you must set several environment variables.

MP_HOSTFILE:	Points to the file containing the list of nodes that you have been assigned to run on by the scheduler.
MP_INFOLEVEL:	Provides an overwhelming amount of debugging information, most of which is often difficult to interpret or misleading. The default level is 1 but should be 0 which is no information. (We have suggested this change to IBM.)
MP_EUILIB:	Tells MPL either to run IP over the switch if set to "ip" or to use direct mode over the switch if set to "us".
MP_PROCS:	Indicates the number of processors the job runs on (should correspond to the number of nodes in the host file).

These variables can be set interactively if you are scheduling interactive time or within a script if you are scheduling batch time. It is crucial that you set the environment variable MP\_HOSTFILE to the file containing the list of nodes the scheduler has allocated to you; otherwise MPL will fail. Here are some examples of a `cs`h batch scripts that will run a MPL programs.

### 2.4.2 MPL Batch Script Example for IP over the IBM Switch

```
#!/bin/csh
# get the job id for this job
set JID = '/usr/local/bin/getjid'
# Run IP over the switch
setenv MP_EUILIB ip
# convert the Scheduler generated SPnodes file to an SWnodes file for
# IP over the switch
cat /sphome/$LOGNAME/SPnodes.$JID | sed 's/sp/sw/' > /sphome/$LOGNAME/SWnodes.$JID
# set SWnodes file the host list for this job
setenv MP_HOSTFILE /sphome/$LOGNAME/SWnodes.$JID
# set the number of processes that you wish to run on
setenv MP_PROCS 64
# set the debugging level (0 is generally recommended)
setenv MP_INFOLEVEL 0
# execute the program with the output redirected to a file
poe /sphome/$LOGNAME/<program_name> >& /sphome/$LOGNAME/job.output
# release the nodes for the next user
srelease $JID
```

### 2.4.3 MPL Batch Script Example for Direct Mode over the IBM Switch

```
#!/bin/csh
# get the job id for this job
set JID = '/usr/local/bin/getjid'
# Run IP over the switch
setenv MP_EUILIB us
# make the scheduler generated SPnodes file the host list for this job
setenv MP_HOSTFILE /sphome/$LOGNAME/SPnodes.$JID
# set the number of processes that you wish to run on
setenv MP_PROCS 64
# set the debugging level (0 is generally recommended)
setenv MP_INFOLEVEL 0
# execute the program with the output redirected to a file
poe /sphome/$LOGNAME/<program_name> >& /sphome/$LOGNAME/job.output
# release the nodes for the next user
srelease $JID
```

*Note to interactive MPL users:*

If you are going to use your interactive time for MPL jobs, you will have to use an MPL host list file so that your MPL job will run on the nodes you have been allocated. If you do not, MPL may try to run your jobs on nodes that you do not have access to, causing the job to fail.

### 2.4.4 MPI

MPI (Message Passing Interface) is a new message-passing system "standard" that has recently been defined by a broadly based group of parallel computing vendors, library writers, and users. The current draft is now in the public-comment stage [3]. Argonne has an implementation of MPI called MPICH [4], which runs on the Argonne SP system. MPI programs can be treated like MPL programs described in the preceding section but have the added capability of being ported to other platforms. The MPICH implementation also has a portable method of starting jobs, called **mpirun**. On the ANL SP system **mpirun** interfaces with the SP scheduler to schedule and start jobs.

### 2.4.5 p4

The p4 system [1], is a portable message-passing system that runs on a wide variety of parallel computers and workstations. It is in use at approximately 200 sites around the world. Existing p4 programs will run unchanged on the SP.

### 2.4.6 p4 Batch Script Example for IP over the IBM Switch

This csh script will produce a procgroup appropriate for IP over the IBM switch.

```
#!/usr/local/bin/tcsh
set JID = '/usr/local/bin/getjid'
cat /sphome/lifka/SPnodes.$JID | sed 's/sp/sw/'>/sphome/lifka/SWnodes.$JID
@ I = 0
touch /sphome/lifka/pg.$JID
foreach i ( 'cat /sphome/lifka/SWnodes.$JID' )
echo "$i          $I          /sphome/lifka/P4/Utest" >> /sphome/lifka/pg.$JID
if ($I == 0) then
@ I = $I + 1
endif
end
/sphome/lifka/P4/Utest -p4pg /sphome/lifka/pg.$JID -p4dbg 10 -p4rdbg 10
```

This example produces the following procgroup file:

```
swnode001      0          /sphome/lifka/P4/Utest
swnode002      1          /sphome/lifka/P4/Utest
swnode003      1          /sphome/lifka/P4/Utest
swnode004      1          /sphome/lifka/P4/Utest
```

### 2.4.7 p4 Batch Script Example IP over the Ethernet

This csh script will produce a procgroup appropriate for IP over the Ethernet between the SP nodes.

```
#!/usr/local/bin/tcsh
set JID = '/usr/local/bin/getjid'
echo "local 0" > /sphome/lifka/pg.$JID
foreach i ( 'cat /sphome/lifka/SPnodes.$JID' )
echo "$i          1          /sphome/lifka/P4/Utest">>/sphome/lifka/pg.$JID
end
/sphome/lifka/P4/Utest -p4pg /sphome/lifka/pg.$JID
```

This example produces the following procgroup file:

```
local          0
spnode001      1          /sphome/lifka/P4/Utest
spnode002      1          /sphome/lifka/P4/Utest
spnode003      1          /sphome/lifka/P4/Utest
spnode004      1          /sphome/lifka/P4/Utest
```

*Note to Interactive p4 users:*

If you are going to use your interactive time for p4 jobs, you will have to create a procgroup file that uses only the nodes you have been allocated. The SPnodes file that the scheduler provides you with contains a list of nodes you can use in your procgroup file. If you submit a p4 program to the scheduler, instead of a job script, it will automatically generate an appropriate procgroup file for you. If you would like to submit a job script to the scheduler you can use the previous csh scripts as a base, which automatically generate an appropriate procgroup file at run-time based on the nodes you are assigned.

### 2.4.8 Task Farm

Task farm users generally use the SP as if it were a large cluster of network connected IBM RS6000 workstations. Binary executables can be submitted to the scheduler, as can shell scripts that run single process jobs on multiple nodes.

## 2.5 Using Temporary Storage on the SP Nodes with the Scheduler

If your program requires that data be moved into and/or out of the /tmp directory on the nodes, it will be important that your program or script perform these moves before and after program execution. This is because you cannot be guaranteed access to specific SP nodes for data staging before the nodes are actually allocated to you or after you have released the nodes using the **sprelease** command. If you request interactive time, you will be able to log into the nodes allocated to you and do your data staging before and after running your program interactively. *NOTE: Once a node is released, all user data in /tmp is removed so that the next user has the same amount of /tmp space available for a job.*

## 2.6 Releasing Resources When Finished

When you are finished with the SP resources that you have been allocated, you will have to release them so that they can be re-scheduled by the system. It is extremely important that you release resources when you are finished with them so that the idle node time is not charged to your CAC. Any allocated time will be charged against your CAC for the resources you requested, whether you use them or not. This is because while the nodes are assigned to you, no one else will have access to them. To return nodes to the scheduler once you are done, you will have to use the **sprelease** command on each of your allocated nodes or with a job ID to release all nodes associated with that particular job ID. **sprelease** will log your release of the node at the time it is issued, remove your access from the node, kill all current processes on the node owned by you, and clear /tmp for the next user. Because /tmp is cleared, it will be necessary to move any important data off the /tmp disk to your /sphome directory *before* releasing the node.

The **sprelease** command can be run interactively by the user, called at the end of a shell script or from a **system** call in your programs. The advantage of embedding it at the end of your scripts or programs is that you will not have to monitor your batch runs to ensure that your resources were released at the correct time. The **sprelease** command offers a big improvement over the sign-up sheet schedulers because nodes can be made available and used again as soon as users are finished using them. In the past, large time-blocks were reserved and held irregardless of how long they were actually used.

## 2.7 Current Queuing Algorithm

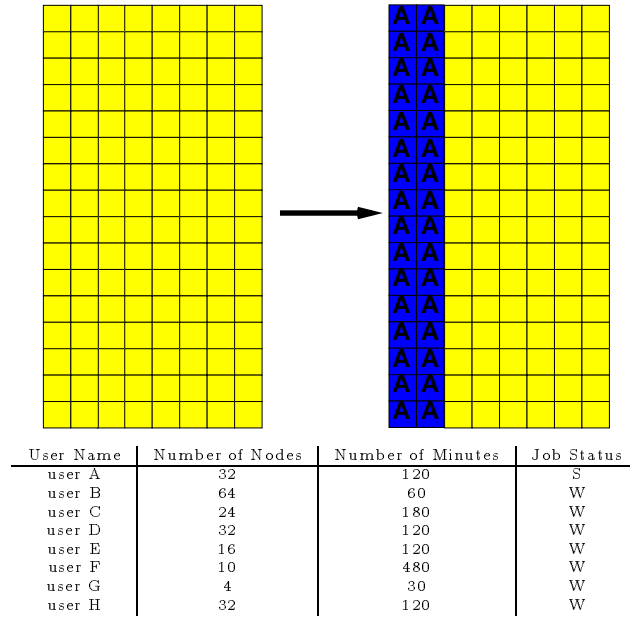
Jobs are currently run in an optimized first-in-first-out (FIFO) fashion. The optimization works by first seeing whether the next job in a particular queue has enough nodes available to start. If so, it is allocated the nodes and started. If there are not enough nodes available for the next job in the queue, the time that job is blocked for is determined by looking at jobs currently on the system and determining how long it will be before enough of them finish so that the next job can start. The scheduler tries to fill this time gap with another job in the queue that will not run longer than this time gap and does not require more nodes than are currently available.

To see resource unit limits that the scheduler uses to classify jobs, use the **spq -h** command. During the time periods when jobs requiring large numbers of resources are started, jobs requiring few numbers of resources will be started if there are no large jobs queued to run. Also notice that jobs requiring very large numbers of resources are considered to be extraordinary events and will be treated as such by the scheduling system.

### 2.7.1 Queuing Example Description

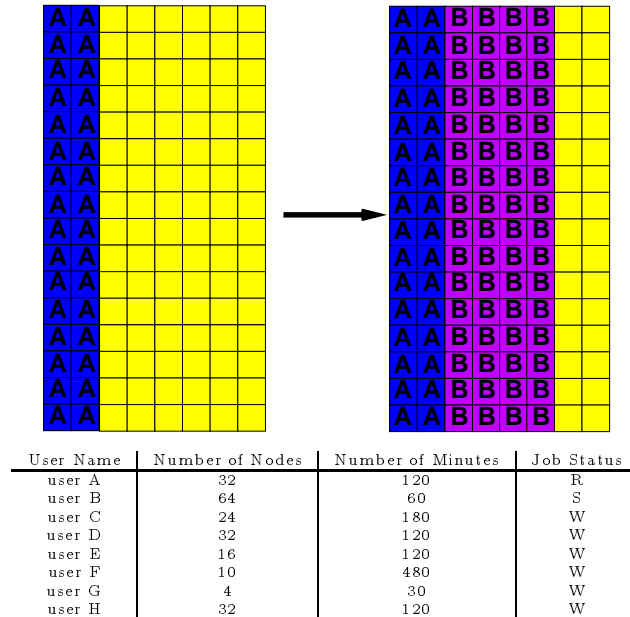
This five-part example shows a typical job load in the queue and shows the machine state before and after each job is started on the system. Each yellow box in the mesh represents a node on the Argonne 128-node SP system. As a job is added to the system the nodes it occupies are changed to a unique color. The Job Status column of the queue table has three states. "W" means the job is waiting to be started, "S" indicates that the job is being started in this part of the example, and "R" means the job is currently running.

### 2.7.2 Queuing Example Part 1



Job A requires 32 nodes for 2 hours. There are 128 nodes available so this job can start.

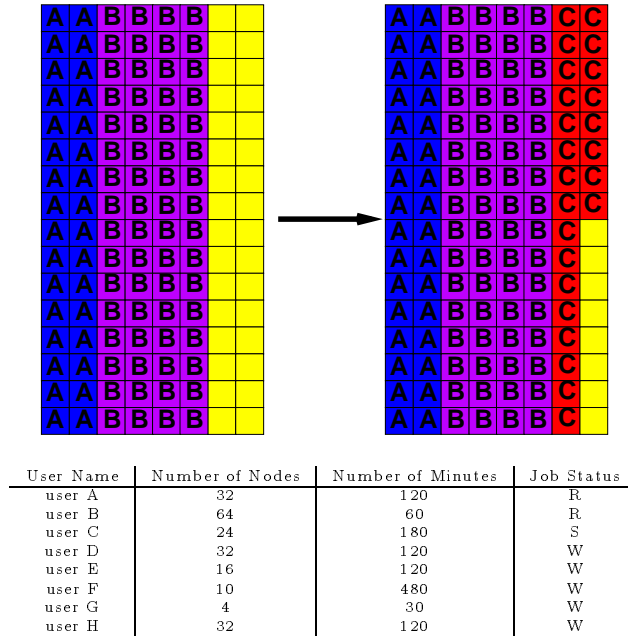
### 2.7.3 Queuing Example Part 2



Job B requires 64 nodes for 1 hour. There are 96 nodes available so this can also start.

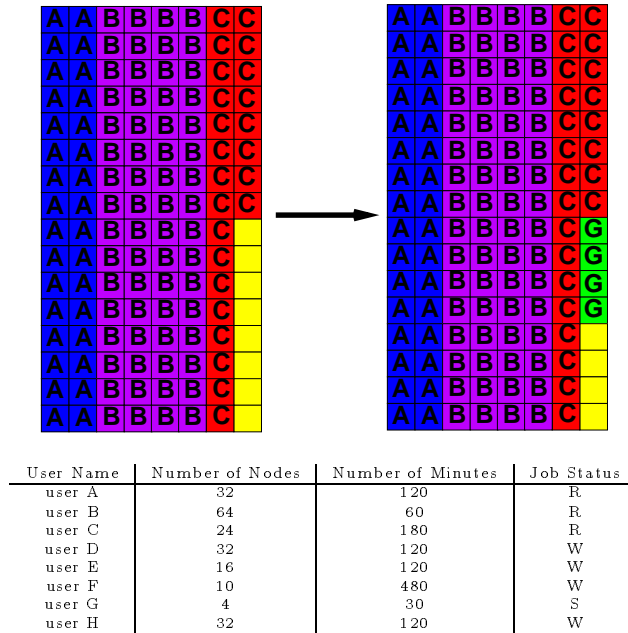


### 2.7.4 Queuing Example Part 3



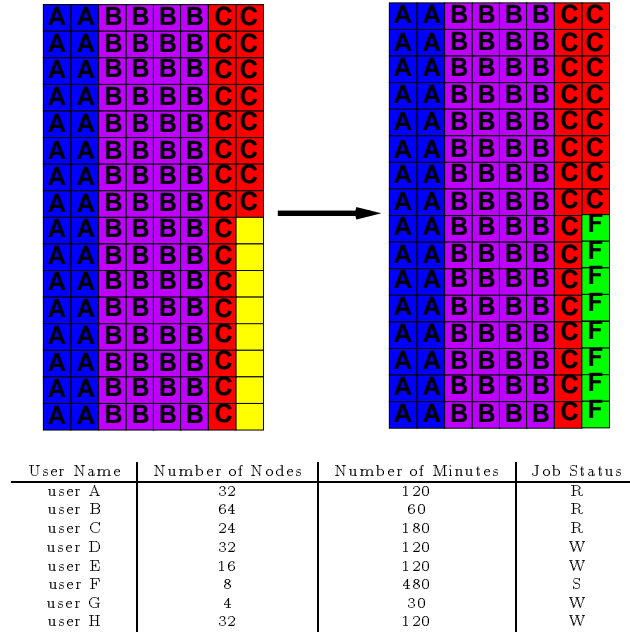
Job C requires 24 nodes for 3 hours. There are 32 nodes available so this can also start.

### 2.7.5 Queuing Example Part 4a



Job D requires 32 nodes for 2 hours. There are only 8 nodes available so this job cannot start. Now the scheduler determines how long it will be before enough nodes currently in use will be available so that job D can start. Job B will finish the soonest, and when it does there will be 8 + 64 nodes available, which is enough for Job D to run. The scheduler now looks for a job that can use the available 8 nodes for 1 hour or less. Jobs E and F need more nodes than are currently free, so they are not candidates. Job G needs 4 nodes for 30 minutes, which is less than the 8 node/60 minute limit, so it is allowed to start.

### 2.7.6 Queuing Example Part 4b



Here is a slightly different scenario assuming the queued job requirements are slightly different. Job D requires 32 nodes for 2 hours. There are only 8 nodes available so this job cannot start. Now the scheduler determines how long it will be before enough nodes currently in use will be available so that Job D can start. Job B will finish the soonest, and when it does there will be 8 + 64 nodes available, which is enough for Job D to run. The scheduler now looks for a job that can use the available 8 nodes for 1 hour or less. Job E needs more nodes than are currently free, so it is not a candidate. Job F needs 8 nodes for 480 minutes, which is greater than the 60 minute limit. Nevertheless Job F is allowed to start because when Job B finishes, it will release 64 nodes, which is 32 more than Job D needs. If Job B released only 24 nodes (with 8 currently available), Job F would not have been allowed to start.

## 3 System Installation and Administration Guide

### 3.1 Installing the Scheduler

This section describes the requirements and steps to install the Argonne SP scheduler. It also goes through the configuration file `Scheduler.config` in detail explaining the various settings and their importance. It assumes a basic understanding of SP system administration.

#### 3.1.1 System Requirements

The Argonne SP scheduling system has several SP system requirements, most of which are normally part of any SP installation. This list is based on what has been needed on Argonne's SP system for the scheduler. It can be used as a reference if things do not work as expected on other systems configured differently.

1. An IBM RS6000 system with at least 20 megabytes of free disk space in a filesystem that can be made NFS mountable to all schedulable resources in the SP complex. At Argonne we use a secondary control workstation, which also has several other administrative purposes. The scheduler does not require a dedicated resource, but it should not be installed on a busy system such as a control workstation or a heavily used compile server.

2. The scheduler has been run under AIX 3.2.4. and 3.2.5. It has not been tested with AIX 4 yet.
3. To enforce *exclusive* access to the SP nodes, the scheduler relies on having an NIS server that all the schedulable resources can rely on for user authentication information. When the scheduler allocates an SP node to a user, it adds that user's username to the SP node's `/etc/passwd` file. This username is used by the NIS server to ensure that the user has access to the SP node.
4. The SP scheduler system is almost entirely written in Perl. It does not use any version-specific features of Perl and is currently using Perl version 4.036. It has not been tested with Perl 5 yet.
5. Tcl and Tk are required if you wish to use the *xspusage* GUI interface. Argonne is currently using versions tcl-7.0 and tk-3.3.

Please send information to [spsupport@ms.anl.gov](mailto:spsupport@ms.anl.gov) about any special features, that are not listed above and are required to get the scheduler working. This will allow us to make future versions of the scheduler more robust.

### 3.1.2 Installation

The following instructions need to be followed by a system administrator with *root* privileges.

1. Find or create an NFS filesystem on the RS6000 that the scheduler is going to be installed on. This filesystem must provide read and write access to all schedulable resources (SP nodes) and to any machine from which users will be able to submit jobs. At Argonne, jobs can be submitted only from the compile servers.
2. Ftp the **scheduler.tar** file to the RS6000 that it is going to be installed on. The scheduler can be obtained by anonymousftp from [info.mcs.anl.gov](http://info.mcs.anl.gov). It is located in the `/pub/sp_scheduler` directory.
3. Untar **scheduler.tar** in the scheduler RS6000 in the NFS-mounted file system created in Step 1. On the Argonne system the scheduler directory is in `/etc/FRAMES`. After untarring the scheduler, the directory **SP\_Scheduler** will be created, which contains the scheduler's contents. At Argonne, the directory containing the scheduler software is in `/etc/FRAMES/SP_Scheduler`. It is important that the directory containing the scheduler be named **SP\_Scheduler**.
4. The scheduler security is built upon Unix file protection. The code is owned by a special group **db\_prot**. Users have **execute** privileges on the scheduler codes owned by **db\_prot**, which can modify scheduler resource files. Users cannot modify the resource files directly, since they are not members of this group.

To create this group, edit the file `/etc/group`. Add the following line to this file:

```
db_prot:*:669
```

This gives the **db\_prot** group the group-id number 669. The user-id number for this group should be 0 or **root**. If group-id number 669 is already taken, you may substitute it with a unique group-id, but you must then also put this number in the **Scheduler.config** file in Step 9 of this installation procedure.

5. Now resource configuration file must be modified to match the system the scheduler is being installed on. To do this, edit the **Resource\_list** file in the **SP\_Scheduler** directory. This file contains a list of the names of the "schedulable resources" in the order that they should be

scheduled if when they are available to run a job. The names of these resources correspond to the hostnames the scheduler will **rsh** commands to. This is a partial example of the Argonne **Resource\_List**:

```
spnode001
spnode002
spnode003
spnode004
spnode005
spnode006
spnode007
spnode008
spnode009
spnode010
spnode011
spnode012
spnode013
spnode014
spnode015
spnode016
spnode017
spnode018
spnode019
spnode020
```

6. Now run the **Configure** script in the Scheduler directory. First **cd** to the **SP\_Scheduler** directory. Then execute the script **Configure**. It will first prompt you for the path to Perl on your system. This is commonly **/usr/local/bin/perl**. It will then prompt you for the path to the scheduler directory. This is the directory you are currently in. On the Argonne system the response would be **/etc/FRAMES/SP\_Scheduler**. This script adds the correct path in the scheduler configuration file to all the scheduler routines that reference it. It will then ask whether you wish to install the **xspusage** GUI interface. If so, it will prompt you for the full path to the Tk program **wish**. This is commonly **/usr/local/bin/wish**. **Configure** will modify all the scheduler codes appropriately for the system it is going to be run on, compile two small C programs that are part of the system, set all the file protections and ownerships, and finally create a resource file that the scheduler will use to keep track of all the schedulable resources.
7. To avoid making SP system users modify their Unix path, create symbolic links from **/usr/local/bin** to the scheduler directory on any system used to access the scheduler. Specifically, create symbolic links for the following scheduler utilities:
  - **spfree**
  - **sphelp**
  - **sppause**
  - **spunpause**
  - **spq**
  - **sprelease**
  - **spsubmit**
  - **spusage**
  - **spwait**
  - **getjid**

8. To guarantee that users have *exclusive* access to the SP nodes that they are allocated through the scheduler, the scheduler adds their username to the `/etc/passwd` file. This username is then used to query the NIS server to allow user access to node. Here is an example of a standard SP node `/etc/passwd` file that allows access to all users known by the NIS server.

```
root::!0:0:::/bin/ksh
daemon::!1:1::/etc:
bin::!2:2::/bin:
sys::!3:3::/usr/sys:
adm::!4:4::/usr/adm:
uucp::!5:5::/usr/lib/uucp:
lpd::!104:9::/:
nobody::!4294967294:4294967294::/:
+::0:0:::
```

In order to allow access to the SP node on a per user basis by the scheduler, this password file must be modified to look like the following.

```
root::!0:0:::/bin/ksh
daemon::!1:1::/etc:
bin::!2:2::/bin:
sys::!3:3::/usr/sys:
adm::!4:4::/usr/adm:
uucp::!5:5::/usr/lib/uucp:
lpd::!104:9::/:
nobody::!4294967294:4294967294::/:
```

This password file should replace the current password file on all the Schedulable resources *except the nodes the MPL Job Manager runs on*. Because of a limitation in its implementation, the MPL Job Manager requires the standard full password file on the nodes it executes on. After creating the restricted-access password file on the nodes copy, it to `/etc/passwd.base`.

```
cp /etc/passwd /etc/passwd.base
```

The `/etc/passwd.base` file is used by the scheduler to remove a user's access from the node. It simply copies `/etc/passwd.base` over `/etc/passwd`, effectively removing the current user's username from the nodes password file.

*Example:* If the scheduler gave a user with the username `lifka` access to an SP node, it would append "+lifka" to the nodes `/etc/password` file. The nodes `/etc/passwd` file then would look like the following.

```
root::!0:0:::/bin/ksh
daemon::!1:1::/etc:
bin::!2:2::/bin:
sys::!3:3::/usr/sys:
adm::!4:4::/usr/adm:
uucp::!5:5::/usr/lib/uucp:
lpd::!104:9::/:
nobody::!4294967294:4294967294::/:
+lifka
```

To remove user Lifka's access from the node the Scheduler simple copies the base password file over the current password file.

9. Several scheduler settings vary from system to system. All configuration settings are contained in the file `Scheduler.config`, which is located in the `SP_Scheduler` directory created in Step 3 of this installation procedure. In order to avoid any confusion, the entire `Scheduler.config`

file is listed below with a brief explanation of each variable or routine, its purpose, and its appropriate settings.

The variable `$DEBUG` is for system administrative purposes. If it is set to `$TRUE`, each piece of the scheduler code will print status messages on the console the scheduler is started from. To disable these messages, set `$DEBUG` to `$FALSE`.

```
$DEBUG = $TRUE;    # Debugging mode ON
$DEBUG = $FALSE;   # Debugging mode OFF
```

The `$Armed` variable is also for system administrative purposes. It was added so that the system administrator could test the Scheduler without actually touching the SP system. If `$Armed` is set to `$FALSE` the Scheduler code will run against the queue of jobs and print messages on the console as to what it would actually be doing. If `$Armed` is set to `$TRUE`, the scheduler will actually schedule and start the queued jobs.

```
#
# Allows the Scheduler to actually "touch" the machine
# (used for testing purposes)
#
```

```
$Armed = $TRUE;
```

During holiday seasons or periods where it is preferable to allow "night" class jobs to run 24 hours day, a special holiday flag can be set. If you wish to start a holiday scheduling time period, the system administrator has to create a file named `HOLIDAY` in the scheduler directory. To end the holiday scheduling period, simply remove this file.

```
#
# Holidays are like Night times all the time
#
```

```
if (-e "$Scheduler_Dir/HOLIDAY")
{
    $Holiday = $TRUE;
}
else
{
    $Holiday = $FALSE;
}
```

The Argonne SP accounting system is currently not available for distribution. For all sites other than Argonne, the `$Scheduler_Accounting` flag should be set to `$FALSE`.

```
#
# Run the ANL/MCS Accounting System ?
#
```

```
$Scheduler_Accounting = $FALSE;
```

In Step 4 of this installation procedure, if you chose a group ID number other than 669 for the scheduler code, you must modify the `$db_prot_gid` flag in the following section of code.

```
#
# File Protection Parameters & Group ID for db_prot
#
```

```
$db_prot_uid = 0;
$db_prot_gid = 669;
```

The `$User_home` variable tells the scheduler where the user's home filesystem is located. On Argonne's SP user's home SP file systems are located in `/sphome/<username>`. If you have a different naming convention for the user's home filesystem, the `$User_home` should be modified accordingly.

```
#
# User's home directory (User NFS mounted file-system that all nodes see)
#

$User_home = "sphome";
```

The `$Scheduler_Host` variable should be set to the Internet address of the computer that the scheduler code runs on.

```
#
# Hostname of the Machine running the Scheduler
#

$Scheduler_Host = "somewhere.mcs.anl.gov";
```

The following section of code controls the scheduler policies. The variables define when different types of jobs can run and ve policy statements for the `-h` options in the various scheduler user utilities. If the scheduler policy does not meet your user community requirements, the following variables can be modified to allow it to do so.

```
#
# These variables describe and define the Night and Day Queues
#

$Day_Time_Def = "Day time is from 9AM to 6PM, Monday through Friday\n";

$Night_Time_Def = "Night time is from 6PM to 9AM, Monday through Friday\n";
$Night_Time_Def .= "and 6PM Friday till 9AM Monday\n";

#
# These are the time limits for the Day & Night Queues
#

$DayJob      = 1920; # Max minutes a Day job can run (32 node/hours)
$NightJob    = 15360; # Max minutes a Night job can run (256 node/hours)
$Queued_Limit = 30720; # Max minutes you can have Queued Waiting
$Week_Night  = 900; # Max minutes on week night

$Limits_Def = "Size Ranges (in Node/Minutes)\n";
$Limits_Def .= "  Day Queue: up to $DayJob";
$Limits_Def .= " AND 9 hours or less.\n";
$Limits_Def .= "  Night Queue : ".$DayJob + 1)." up to 63 node/hours\n\n";
$Limits_Def .= "After a CAC has $Queued_Limit Node/Minutes scheduled per\n";
$Limits_Def .= "queue, additional jobs are \"Held\" by the system till\n";
$Limits_Def .= "the total Node/Minutes that CAC has requested in that queue\n";
$Limits_Def .= "fall below $Queued_Limit Node/Minutes. At that point Held\n";
$Limits_Def .= "jobs are returned to \"Waiting\" status.\n";
$Limits_Def .= "\nThis allows a user to submit many jobs at one time without\n";
$Limits_Def .= "denying access to other active users.\n";
```

Most scheduler code requires the following information. The only option that may need changing is the `$Number_of_Nodes` flag, which corresponds to the number of schedulable SP nodes in the SP system.

```
#
# these are variables used in most of the Scheduler daemons
#

$Number_of_Nodes = 128; # number of nodes in the system
$LockWait = 1200; # daemons wait 20 minutes for file lock

%days_in_Month = ("January", "31", "February", "28", "March", "31",
                  "April", "30", "May", "31", "June", "30", "July", "31",
                  "August", "31", "September", "30", "October", "31",
                  "November", "30", "December", "31");

@Months = ("January", "February", "March", "April", "May", "June",
           "July", "August", "September", "October", "November", "December");
```

After the declaration of all the essential scheduler variables there are several subroutines which are used by several pieces of the scheduler code. They follow the following comment lines.

```
#
# These are special Scheduler Subroutines
#
```

The first two, `AmIRunning` and `GetMyName`, are used to ensure that only one of each of the scheduler daemon processes is running on the scheduler host computer.

```
#
# Subroutine to ensure only one copy of a daemon is running
#

sub AmIRunning
{
    $Process_Name = $_[0];
    $occurrences = 0;
    $hostname = '/bin/hostname';
    chop $hostname;
    # Make sure you are running on the correct machine
    if ($hostname eq $Scheduler_Host)
    {
        # Make sure your not allready running
        open(PS,"/bin/ps auwx|");
        while (<PS>)
        {
            if (/ $Process_Name/) { $occurrences++; }
        }
        if ($occurrences > 1)
        {
            if ($DEBUG) {print "DEBUG> Found $occurrences of this program running\n";}
            return 1;
        }
        return 0;
    }
    else

```



```

    {
        print "This program only runs on $Scheduler_Host\n";
        return 1;
    }
}

```

```

#
# Returns the program name without the path
#

```

```

sub GetMyName
{
    $Full_Name = $_[0];
    while ($Full_Name =~ m#/#) {$Full_Name = "$'";}
    return ($Full_Name);
}

```

The routine `IsA_Node` is used to ensure that when a user issues a `getjid` or `sprelease` code, that is done from a valid schedulable resource. It assumes that nodes from which these commands could be run have the same names as those in the `Resource_List` file that was created in Step 5 of this installation procedure. If an SP node responds as a different hostname from that listed in the `Resource_List`, you can modify the hostname conversion section of this subroutine to convert its hostname to its corresponding `Resource_List` name.

```

#
# Ensures that the Node name passed is a scheduable resource
#

```

```

sub IsA_Node
{
    $Check = $_[0];
    #
    # perform any special hostname conversions here
    #
    if ($Check eq "mercury") { $Check = "spnode001"; }
    if ($Check eq "venus")   { $Check = "spnode017"; }
    if ($Check eq "earth")   { $Check = "spnode033"; }
    if ($Check eq "mars")    { $Check = "spnode049"; }
    if ($Check eq "jupiter") { $Check = "spnode065"; }
    if ($Check eq "saturn")  { $Check = "spnode081"; }
    if ($Check eq "uranus")  { $Check = "spnode097"; }
    if ($Check eq "neptune") { $Check = "spnode113"; }

    open (RESOURCES,"$Scheduler_Dir/spusage |");
    while (<RESOURCES>)
    {
        chop;
        ($node, $avail, $user, $JID, $StopTime) = split;
        if ($node eq $Check) { return ($node); }
    }
    close(RESOURCES);
    return("");
}

```

The subroutine `UserInfo` simply returns the effective user ID for a given user. This routine should not have to be modified.

```
#
# Gets the effective uid for the user
#
```

```
sub UserInfo
{
    @password = getpwuid($>);
    $username = $password[0];
    $username =~ s/,.*//;
}
```

The subroutine `TimeStamp` simply returns the current time, which is used to generate a job ID and also to determine whether it is daytime or nighttime. Several of the Scheduler programs use this routine.

```
#
# Sets the current time and JID for a given job
#
```

```
sub TimeStamp
{
    ($sec,$min,$hour,$mday,$mon,$year,$yday,$isdst) = localtime();
    $mon++;
    $msec = '$Scheduler_Dir/GenJID';
    chop($msec);
```

```
    # put in leading zeros as needed
    if ($mon < 10) { $mon = "0".$mon; }
    if ($mday < 10) { $mday = "0".$mday; }
    if ($hour < 10) { $hour = "0".$hour; }
    if ($min < 10) { $min = "0".$min; }
    if ($sec < 10) { $sec = "0".$sec; }
```

```
    $TimeNow = "$mon$mday$hour$min";
    $logtime = "$mon$mday$hour$min$sec";
    $ClockTime = "$hour$min$sec";
    $Midnight = "240000";
    $DayStart = "090000";
    $NightStart = "180000";
    $JID = $mday.$hour.$min.$sec.$msec;
    $Sunday = 0;
    $Monday = 1;
    $Tuesday = 2;
    $Wednesday = 3;
    $Thursday = 4;
    $Friday = 5;
    $Saturday = 6;
    $Today = $yday;
```

```
    # Determine if it is Night or Day
    if (($Today == $Saturday) || ($Today == $Sunday))
    {
        $DayTime = $FALSE;
        $NightTime = $TRUE;
    } #if its a weekend
```

```

if (($Today >= $Monday) && ($Today <= $Friday))
{
    if (($ClockTime ge $DayStart) && ($ClockTime lt $NightStart))
    {
        $DayTime    = $TRUE;
        $NightTime = $FALSE;
    }
    else
    {
        $DayTime    = $FALSE;
        $NightTime = $TRUE;
    }
} #if its a weekday

# Override normal settings if its a Holiday
if ($Holiday)
{
    $DayTime    = $FALSE;
    $NightTime = $TRUE;
}
}

```

The `EndTime` subroutine determines when a job started now will finish, based on the number of minutes it has requested.

```

sub EndTime
{
    $Time = $_[0];
    # calculate minutes from days & hours
    $req_day = int( $Time / 1440.0 );          # the number of minutes in a day
    $req_hour = int( ( $Time % 1440 ) / 60 ); # number of hours
    $req_min = $Time % 60;
    # split the current time
    $now = $TimeNow;
    $now =~ /(\d\d)(\d\d)(\d\d)(\d\d)/;
    $now_mon  = $1;
    $now_day  = $2;
    $now_hour = $3;
    $now_min  = $4;
    # calculate stop times
    $end_mon = int($now_mon);
    $end_day = int($now_day) + $req_day;
    $end_hour = int($now_hour) + $req_hour;
    $end_min = int($now_min) + $req_min;
    # handle roll-overs
    if ($end_min > 59)
    {
        $tmp = $end_min;
        $end_min = $tmp % 60;
        $end_hour += int( $tmp / 60.0 );
    }
    if ($end_hour > 23)
    {
        $tmp = $end_hour;
        $end_hour = $tmp % 24;
    }
}

```

```

    $end_day += int($tmp / 24.0);
}
while ($end_day > $days_in_Month{@Months[$now_mon-1]} )
{
    $end_day -= $days_in_Month{@Months[$now_mon-1]};
    $end_mon++;
    if ($end_mon == 13) { $end_mon = 1; }
}
# now format the EndTime
if ($end_min < 10) { $end_min = "0".$end_min; }
if ($end_hour < 10) { $end_hour = "0".$end_hour; }
if ($end_day < 10) { $end_day = "0".$end_day; }
if ($end_mon < 10) { $end_mon = "0".$end_mon; }
return ($end_mon.$end_day.$end_hour.$end_min);
}

```

The subroutines `TimeLimit`, `TodayAt`, and `TomorrowAt` are used to determine how long a job can run, based on the current time and day of the week.

```

sub TimeLimit
{
    # This subroutine returns a run-time limit in the form MMDDHHMM
    $JobClass = $_[0];

    # Monday - Friday behave the same
    if (($Today >= $Monday) && ($Today <= $Friday))
    {
        if (($ClockTime >= $DayStart) && ($ClockTime <= $NightStart))
        {
            $DayLimit = &TodayAt("1800");
            $NightLimit = "000000"; # Don't start Night jobs on Weekdays
        }
        else
        { # 6pm till Midnight
            if (($NightStart <= $ClockTime) && ($ClockTime < $Midnight))
            { # Handle Monday - Thursday nights
                $DayLimit = &TomorrowAt("1800");
                $NightLimit = &TomorrowAt("0900");
            }
            else
            { # handle early Mornings
                $DayLimit = &TodayAt("1800");
                $NightLimit = &TodayAt("0900");
            }
        }
    }
    # if Monday - Friday

    # Today is Saturday or Sunday
    if (($Today == $Sunday) || ($Today == $Saturday))
    {
        $DayLimit = &TomorrowAt("1800");
        $NightLimit = &TomorrowAt("0900");
    } # if today is Saturday or Sunday

    # If this is a Holiday

```

```

if ($Holiday)
{
    $DayLimit = &TomorrowAt("0900");
    $NightLimit = &TomorrowAt("0900");
} # if this is a Holiday period

if ($JobClass eq "DAY") { return($DayLimit); }
if ($JobClass eq "NIGHT") { return($NightLimit); }
}

sub TodayAt
{
    $time = $_[0];
    return($mon.$mday.$time);
}

sub TomorrowAt
{
    $time = $_[0];
    if (($Today >= $Sunday) && ($Today < $Friday))
    { $manana = int($mday) + 1 };
    if ( $Today == $Friday ) { $manana = int($mday) + 3 };
    if ( $Today == $Saturday ) { $manana = int($mday) + 2 };
    $Tmon = int($mon);
    $Over_days = $manana - $days_in_Month{@Months[$mon-1]} ;
    if ($Over_days > 0)
    {
        $Tmon++;
        if ($Tmon == 13) { $Tmon = 1; }
        $manana = $Over_days;
    }
    if ($Tmon < 10) { $Tmon = "0$Tmon"; }
    if ($manana < 10) { $manana = "0$manana"; }
    return($Tmon.$manana.$time);
}

```

The **CheckNode** routine had to be written to due to a bug in the SP Resource Manager. The **switch responds** flag for a given node may respond positively even though it is not possible to run a job that requires IP over the switch. **CheckNode** tests the node by rsh'ing to the node's ethernet address a ping of the node's switch address. If either do not respond the node will be marked down and taken out of service so that the Scheduler does not try to start new jobs on it. This routine assumes that the switch host name for any given SP node is **swnode** + the nodes number, for example, **swnode001**, **swnode065**, or **swnode128**. If this does correspond to the naming conventions on your local SP, this routine will need to be modified accordingly.

```

sub CheckNode
{
    # this will always be in the form "spnode###"
    # (IsA_Node does appropriate name conversions for various
    # system configurations

    $Node = $_[0];

    #
    # Notice: Non-ANL systems need to modify this routine.  On the ANL system

```

```

# since the Resource Manager does not provide reliable information
# on "switch-responds" we rsh to the Ethernet address a ping of its
# own switch address. This will verify that both network interfaces
# are up and can run any type of communications a job may require.
# On ANL's system the Ethernet address is "spnode###" and the corresponding
# switch address is "swnode###".
#

# get just the node number to append to the string "swnode"
$Node_num = $Node;
$Node_num =~ s/spnode//;

$Response='$timeout 10 /usr/bin/rsh $Node /etc/ping -c 2 swnode$Node_num';
return ($Response);
}

```

The last routine `locksmith`, was developed at Argonne to cleanly perform the locking and unlocking of the files. It is used by the scheduler code to lock and unlock the `SP_Resource` file. It should not have to be modified.

```

sub locksmith
{
    local($timeout, $lockfile, $command) = @_;

    if ($command eq 'release')
    {
        unlink($lockfile) || warn "Could not unlink $lockfile: $!\n";
        return 1;
    }
    elsif ($command eq 'release_maybe')
    {
        unlink($lockfile);
        return 1;
    }
    elsif ($command ne 'lock')
    {
        die "Invalid command '$command' in locksmith\n";
    }
    local($endtime) = time + $timeout;

    #
    # If there is a locksmith_RS6000, then we are
    # on a machine where we can't do a syscall
    #

    if (-x "$Scheduler_Dir/locksmith_RS6000")
    {
        require 'errno.ph';
        local($rc, $err, $ret, $errstr, @errstr);

        while (time < $endtime)
        {
            open(DOLOCK, "$Scheduler_Dir/locksmith_RS6000 $lockfile |");
            $ret = <DOLOCK>;
            close(DOLOCK);

```

```

($rc, $err, @errstr) = split(/\s+/, $ret);
$errstr = join(" ", @errstr);

if ($rc >= 0)
{
    return 1;
}
elsif ($err != &EEXIST)
{
    die "Error in open: $errstr\n";
}
sleep 1;
}
}
else
{
    local($fd);
    require 'sys/file.ph';
    require 'sys/fcntl.ph';
    require 'syscall.ph';

    while (time < $endtime)
    {
        $fd = syscall(&SYS_open, $lockfile, &O_CREAT | &O_WRONLY | &O_TRUNC |
        &O_EXCL, 0644);
        if (!$fd)
        {
            warn "syscall failed: $!\n";
        }
        elsif ($fd >= 0)
        {
            open(FH, ">&$fd") || die "perl open failed: $!\n";
            print FH time . "\n$$\n";
            close(FH);
            return 1;
        }
        sleep 1;
    }
}
return undef;
}

```

10. The last step in the scheduler installation is to run **Scheduler\_Startup** on the scheduler's host computer. If you wish to have this happen after any reboot, you can modify the `/etc/rc.local` file on the workstation that the scheduler runs on.

*NOTE: It is important to move the scheduler log files off the scheduler's file system on a regular basis to ensure that they do not fill the file system. This can easily be done by using a cron job.*

## 3.2 Modifying the Scheduler for a Specific Environment

The scheduler can be modified to handle any message-passing library. The only routine that needs to be modified is `spsubmit`. If special job setup is required for a particular message-passing library, `spsubmit` can be modified to handle it by adding code to it that works much as the current code

currently does for MPL and p4 jobs. The `spsubmit` routine can be used as-is if users are told to choose the "task farm" option when submitting jobs and submitting a shell script, which does all the appropriate setup for the given message-passing system.

### 3.3 System Administration Tools

Several utilities were designed to make administration of the scheduler and the SP easier. `Scheduler_Startup` simply starts all the necessary scheduler processes. The two utilities are `Scheduler_ON` and `Scheduler_OFF` are used to enable graceful interrupts. Sometimes it is necessary to stop the scheduler for urgent SP system administration. In order to stop the scheduler without corrupting the status of the queue or the currently running jobs, `Scheduler_OFF` signals all the scheduler processes so that they can complete any task they are currently working on and then stop gracefully. `Scheduler_OFF` does not actually kill the scheduler processes; it just pauses them until they are either killed by hand or restarted by signaling them with `Scheduler_ON`. Another utility, `sprelease`, is the same as the user command only it is more powerful for the administrator logged in as `root`. As `root`, `sprelease` will allow a system administrator to take a job off any particular node or off all the nodes it is running on in the case of parallel program. It essentially is the scheduler's version of the Unix `kill` command.

The `service` command has several functions. It provides a clean way for the administrator to take an set of nodes out of service and still allow the scheduler to schedule jobs that can run on the remaining nodes. Its basic syntax is

```
service <when> <reason> first_node last_node
```

The `<when>` field can either be `-n` for service immediately and inform the current user of the node by email that it had to be taken out of service or `-s` or service the node the next time it becomes available. Argonne has several reasons for "servicing" nodes, but these can be modified renamed to better suit other environments. The choices that are currently available are as follows.

- h prints a list of the various flags
- f puts nodes back in service
- u mark the nodes down for service or basic maintenance
- g mark the nodes available for general use
- c mark the nodes down for CAVE use
- d mark the nodes down for a demonstration

If you decide to modify these, you will have to modify `xspusage` accordingly so that the correct message is displayed to the users.

The last utility is `Clear_SP_Resources`. It is used to regenerate a clear resource file after a long period of downtime when you would like the scheduler to start as if it were the first time it was turned on.

The "scheduler message" is a way for the administrators to display current information to the users about the current status of the machine for example after a compiler upgrade. This message is displayed by `spq` and `spsubmit` so that users are sure to see it. To modify this message, simply edit the file `Scheduler_message` file in the scheduler directory.

Finally, there are log files generated by each of the scheduler daemons that contain information about what they are doing during every iteration of their execution. These are useful if the administrator wants to see why the scheduler treats various queues of jobs the way that it does and to verify that it is working the way it should.



## 4 Scheduler Architecture

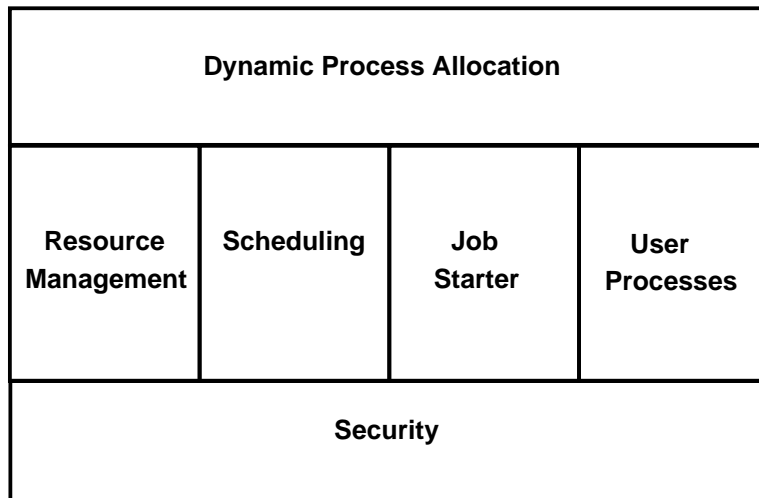
### 4.1 Components of a Scheduling System

Before we began to write a new scheduler, a lot of thought went into what exactly it was a scheduling system should provide. There were three basic goals that almost any scheduling system strives for: fairness; simplicity (ease of understanding); and efficient use of the available resources. These three goals are obviously in conflict, so there had to be some compromise that would make the users happy. After a fair amount of research a list of features making up the "Ultimate Scheduler" was developed. This "Perfect Scheduler" would

- provide optimum utilization (e.g., schedule I/O bound and CPU bound jobs together),
- be fair,
- support different job classes (e.g., interactive vs. batch),
- support various message-passing libraries,
- use static or dynamic partitioning of the machine,
- utilize time or space slicing, gang scheduling, or sign-up sheet mechanisms,
- schedule different computation models (task farm vs. parallel processing),
- manage other system resources (e.g., I/O subsystems),
- provide priority scheduling for special jobs.

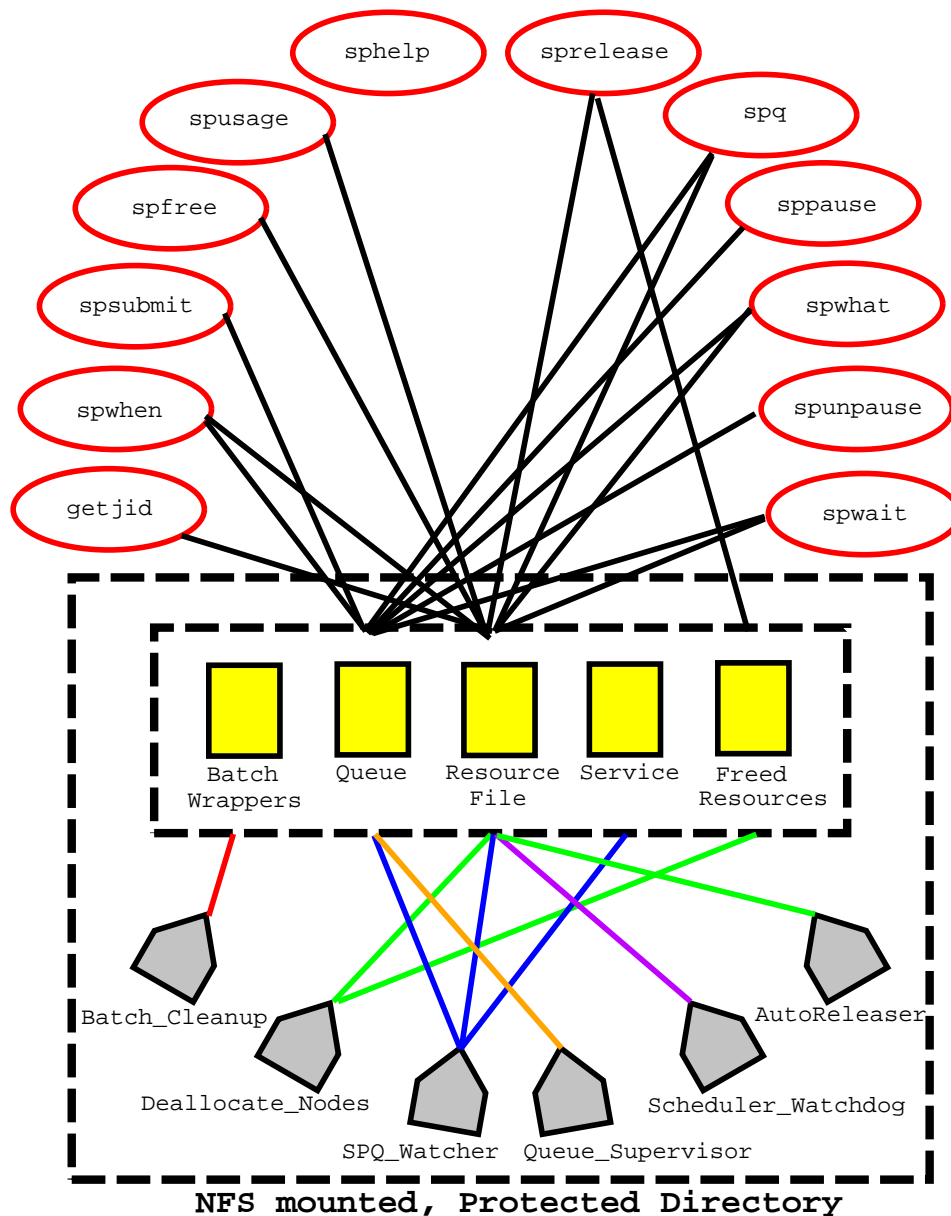
Several of these items really depend upon how the users of a machine expect to be able to use it. Several very nice scheduling systems available today that address these issues. A few of the more popular are DQS from Florida State, Condor from University of Wisconsin, IBM LoadLeveler, and NQS. The problem with these systems is that they all primarily focus on managing multiple queues of nonparallel jobs for networks of workstations. They were developed during the "free supercomputing" movement, not too long ago when high-end workstations connected by fast networks could provide as much computational power as the supercomputers of the day at a fraction of the cost. Many of these scheduling systems do more than scheduling. The following diagram shows the main pieces of a complete scheduling system. Several of the available scheduling systems have implemented the various pieces of this diagram in a tightly coupled fashion. This greatly reduces the extensibility of the system. For this reason a scheduling system that would meet the Argonne goals addresses only "Scheduling" and attempts to get the other pieces from either the machine vendors or other developers wherever possible.

The following diagram illustrates the various pieces of a full scheduling system. This scheduler only performs the "scheduling" task.



## 4.2 SP Scheduler Design

The scheduler is made up of a series of Perl daemons and user interface programs that monitor several shared directories. Those directories are accessible to the entire SP complex. They contain information on the current state of the machine and queue of jobs and update a single resource file that stores this information. The user interface programs are discussed in Section 2.2 of this document. There are currently six Perl daemons, as shown in the following diagram.



These scheduler daemons have the following jobs:

**Batch\_Cleanup:** This is a simple cleanup utility that removes batch scripts from the `Batch_wrappers` directory when they are no longer on the system.

**Deallocate\_Nodes:** Watches for jobs that have been released and then clears the resources they used, verifies they are still operational, and returns them to the "free pool".

**SPQ\_Watcher:** Monitors the job and service queues and starts `Allocate_Nodes` if there is a job that needs starting.

**Queue\_Supervisor:** Ensures that no user has more jobs queued than are allowed in the current queuing policy.

**Scheduler\_Watchdog:** Monitors the SP resources to ensure that they are operational. If a SP node is down, `Scheduler_Watchdog` will take it out of service; if a node comes back up, this daemon will put it back in service.

**AutoReleaser:** Removes jobs from the system that have used all the time they requested.

### 4.3 Sample Flow of Execution

The following flow-of-execution example is provided to help clarify the roles of the various pieces of the scheduler system.

1. User submits a job with `spsubmit`.
2. The `Queue_Supervisor` ensures this job is not one of several requiring more resources and time than the queue limit.
3. Job is "noticed" by the `SPQ-Watcher` daemon.
4. `SPQ-Watcher` Starts `Allocate_Nodes` program to start the job.
5. User is granted access to a group of nodes, and their job is `rsh'd` to the first node in that group of scheduled nodes.
6. The `AutoReleaser` monitors the resource file to ensure the job has not run out of time.
7. When the job has finished it "spreleases" its nodes.
8. `sprelease` puts tokens in the `Freed_Resources` directory to notify the `Deallocate_Nodes` daemon.
9. `Deallocate_Nodes` daemon clears and checks the released nodes and returns them to the `Free_Pool`.
10. The `Batch_Cleanup` daemon realizes the job has now finished, so it removes the jobs batch script from the `Batch-wrapper` directory.
11. The `Scheduler_Watchdog` daemon "patrols" the machine looking for nodes in need of service or that have been repaired and should be put back in the "free-pool".

## References

- [1] Ralph Butler and Ewing Lusk. User's guide to the p4 parallel programming system. Technical Report ANL-92/17, Argonne National Laboratory, October 1992.
- [2] William Gropp and Ewing Lusk. Scalable Unix tools on parallel processors. In *Proceedings of the Scalable High-Performance Computing Conference, IEEE*, pages 55–62, 1994.
- [3] Message Passing Interface Forum. Document for a standard message-passing interface. Technical Report Technical Report No. CS-93-214 (revised), University of Tennessee, April 1994. Available on **netlib**.
- [4] William D. Gropp and Ewing Lusk. A test implementation of the MPI draft message-passing standard. Technical Report ANL-92/47, Argonne National Laboratory, December 1992.
- [5] William D. Gropp, Ewing Lusk, and Steven C. Pieper. Users guide for the ANL IBM SP1. Technical Report ANL/MCS-TM-198, Argonne National Laboratory, October 1994.