

Preemption and Priority Calculation in Production Job Data

Martin W. Margo, Kenneth Yoshimoto, Patricia Kovatch

San Diego Supercomputer Center, University of California, San Diego
{mmargo, kenneth} @sdsc.edu
{pkovatch} @utk.edu

Abstract.

In order to offer on-demand computing services as well as improve overall job throughput, the San Diego Supercomputer Center has implemented preemption on its production supercomputers. Preemption allows time critical applications to run as needed and also allows more jobs to be processed through the queue since the jobs can backfill in several smaller discrete blocks of time rather than in one contiguous block. With our local home grown scheduler, Catalina[4], we implemented preemption to enable these new usage scenarios. We explore the impact of the job expansion factor and utilization when preemption and job priority are taken into account on both long-running and large-scale jobs. We simulated real production log using a simulator to investigate preemption impact to scheduling metrics.

1. Introduction

Preemption allows a high priority job to start immediately by suspending the currently running job. For the current study, we define preemption as the suspension of job execution. This involves halting the job, allowing it to stay resident on compute resources while another job runs on those resources, and then resuming execution of the suspended job. Preemption has been studied extensively [1,2,3] but it hasn't been used widely on production supercomputers.

We were interested in preemption to improve our service to two groups of users: users wanting real-time turnaround and users wanting faster overall service. Several users wanted to have timely turn-around time on their jobs. For instance, some biomedical researchers in the middle of an experiment needed to have results of data immediately since it would help them decide how next to proceed in the experiment. They needed real-time, on-demand access to the results of their jobs. They are willing to be charged at a higher rate than normal to get access to this timely service.

Other users would like their jobs to finish faster. Often schedulers cannot schedule idle nodes since no lower priority jobs can complete in time before the higher priority job is scheduled to run. At SDSC we give higher priority to long-running, wide (large numbers of nodes) jobs to encourage capability parallel applications. This means that much of the machine can sit idle while the machine is draining for a full-machine run. This is in conflict with our dual goal of aiming for high utilization of our supercomputing resource. If users are willing for their jobs to be preempted, possibly by implementing application-level checkpointing, then their jobs can fit into the queue in smaller time slices. This is even more attractive if they are charged less for their jobs that may be preempted but will probably finish more quickly than waiting for a con-

tiguous block of time. If these jobs can backfill into more time slices then the overall system utilization and throughput is increased.

To meet these goals, we implemented preemption in our local scheduler, Catalina.

2. The Catalina Scheduler

The Catalina scheduler was developed to serve as a maintainable, extendable external scheduler for IBM's LoadLeveler[5] the Portable Batch System (PBS)[6] and other resource managers. Support of the GridForum Advance Reservation API[7] was one of the goals of the development effort. Python was chosen as the development language, to encourage readability, with some functions written in C. Catalina is a full-featured scheduler written in a brief 10,000 lines. The license is free for government and non-profit users. It has been used in production at the San Diego Supercomputer Center (SDSC) since 2001.

Catalina is similar to the Maui Scheduler[8] in many ways. It is a single queue, priority-based scheduler. Both schedulers consider all the jobs on a machine in a single queue and schedule them based on a priority calculation. It can accept priorities on jobs, users or groups. Catalina has ten separate factors that it uses to make a priority calculation. It calculates the expansion factor for how long a job is waiting and adjusts the priority on the job so it won't starve. We will discuss this further in the priority section.

Both schedulers support reservations made in advance of the event and standing or recurring reservations. They support restriction of jobs from nodes for set time intervals based on job characteristics. Catalina binds jobs to nodes and reserves nodes for jobs. Both schedulers allow specification of recurring reservations. When a system reservation is made, jobs that will complete before the system reservation starts are scheduled to run.

Backfill is another feature that both schedulers implement though they implement it differently. They effectively run lower priority jobs as long as they don't delay higher priority jobs. This keeps the processors busy until the right number of nodes is available for a larger job with higher priority. Catalina only does conservative backfill, so no jobs are delayed. Maui defaults to aggressive backfill, where the top job is not delayed, but others may be.

Both schedulers support preemption. A job can be preempted if an administrator requests it or if a job is marked as preemptible. We will discuss this in more detail in the next section.

Catalina does not natively support several features that Maui (version 3.0.3.7) does including fairshare, real-time job statistics tracking and workload profiling. However, with other SDSC-developed scripts and IBM's LoadLeveler and PBS we get this functionality. Fairshare or setting priorities based on a group's usage, is supported at SDSC through a job filter script. When a job is submitted to the batch queue on SDSC's production supercomputers, the job filter script checks to see if a user has enough allocation for the job to finish in the amount of time requested. Job statistic information such as the job expansion factor and queue wait time as well as the profile information on the workload are saved in Catalina history files but are not available in real-time. This includes information such as how many 128-node jobs ran in the last year.

Catalina has several distinguishing features. One of them is the implementation of shortpools. These are virtual sets of nodes guaranteed to be available after a configurable lag time. For example, Catalina can guarantee that 32 nodes will be available for the first high-priority job to enter the queue, after a maximum of 8 hours queue wait time.

Catalina also has arbitrary Python code for select jobs allowed to use a reservation. For example, we can configure Catalina reservations to allow jobs to use a particular reserved set of resources if those jobs have a any of a variety of characteristics. For example, job size, job account, username, node name, memory characteristics, etc.

Another feature of Catalina is user-settable reservations. This feature allows an unprivileged user to create a reservation, subject to site policies. This allows application-level metaschedulers to create reservations without requiring administrative privileges on the local cluster. We demonstrated this feature with the Grid Universal Remote (GUR)[9], where no centralized metascheduler was needed to create co-scheduled reservations between different sites on a grid.

Catalina is consistent with the GridForum Advance Reservation API. It supports the basic tasks of creating a reservation modifying a reservation, binding a reservation to a job and canceling a reservation.

3. Implementation of Preemption in Catalina

There are two ways of preempting a job. In the first way, a system administrator executes a command manually. The system administrator uses privileges to execute the command `llpreempt`. The `llpreempt` command is part a LoadLeveler command that bypasses Catalina altogether. However, Catalina is able to detect the manual preemption commands via changes in the LoadLeveler state. Catalina can resume the preempted jobs if the priority calculations favor this preempted job. We will discuss the priority calculations later in the paper.

In the second way, Catalina automatically preempts the job if it's tagged as preempting and/or preemptible. The user needs to mark a job as preemptible. This can be done by the user at submit time or SDSC's job filter script can tag a job automatically.

Catalina Preemption Algorithm [res. = reservation]

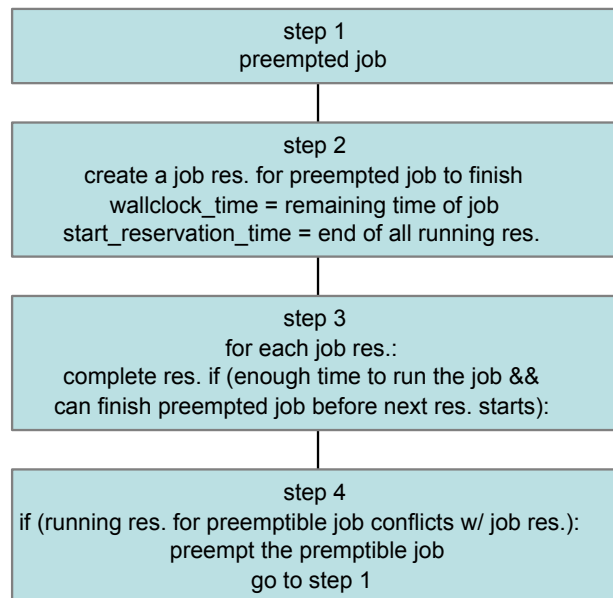


Fig. 1. Catalina's preemption algorithm

When a preemption request is made, LoadLeveler suspends the processes on the node, the job is marked as preempted in LoadLeveler and Catalina and the resources are now free to run the new higher priority jobs.

The preemption process is depicted in detail in Figure 1. It illustrates Catalina's algorithm to schedule preempted and preempting jobs as well as regular batch jobs. In step 2, a reservation is created to finish the preempted job in step 1. The new reservation wall clock time is the remaining time of the job being preempted. The new reservation starting time is the end of all currently set reservations. When Catalina reaches the reservation starting time, it will check to see if the reservation can run to completion before starting the reservation. If not, it will start the job and then preempt again.

Catalina’s preemption rules are illustrated in Table 1.

A job labeled “preempting” may preempt a job labeled “pre-emptible”
A job may preempt a running job only if the preempting job has a higher priority
A job may be preempted only if both it and the preempting job can be completed before the next system or user reservation
On any given node, there may be only one preempted job at a time
If a running job has less remaining wall clock time than PREEMPT_MIN_RUNTIME seconds (default 1800) then it will not be preempted

Table 1. Catalina preemption rules

Catalina’s preemption implementation is user-friendly. In order for a job to be pre-emptible or preempting, the user has to voluntarily modify his or her Job Control File (JCF). Moreover, the rules in Table 1 are conservative because Catalina only allows higher priority job to preempt others, one preemption at a time on a node, and there is a reasonable grace period where a job wouldn’t get preempted if the time remaining is short enough.

To use preemption in Catalina, a user needs to modify the JCF LoadLeveler jobfile. Here, we illustrate Catalina preemption using DataStar system.. DataStar is an IBM p-Series machine, with 272 Power4+, 8-cpu nodes and 6 Power4+ 32-cpu nodes. IBM’s Federation Switch is used to provide high performance communication between nodes. We will describe two batch jobs, job A and job B.

Suppose the owner of job A planned to run a non-urgent long-running job. The job is not time sensitive, therefore the user doesn’t have a strong preference of when the job should finish. The user might want to voluntarily mark the job ‘preemptible’. When a job is marked as preemptible, another job with higher priority may preempt it according to the rules in Table 1. The owner of job A needs to declare the following statement to the JCF jobfile:

```
#@ comment = Catalina_preemptible=Y;
```

Suppose the owner of job B needs to run an urgent batch job. The user might choose to submit a preempted job, even if it costs more. To do this, the user has to mark the job 'preempting'. A job with this mark can automatically preempt other jobs with lower priorities as described by the rules in table 1. The owner of job B needs to declare the following statement on the JCF jobfile:

```
#@ comment = Catalina_preempting=Y;
```

Preemption is a single side-track on which to temporarily hold jobs while a higher priority job comes through. One level of preemption should allow the new jobs to use the node resources without interfering with the previous job.

4. Priority Calculations

Kettimuthu[1] made assumptions that all scheduled jobs have equal priority. In production supercomputers this might not be the case. For example, on SDSC's DataStar (DS) supercomputer, wide jobs have higher priority than narrow jobs. Wide jobs are batch jobs that run on a large number of nodes in parallel, vice versa, narrow jobs are ones that run on a small number of nodes. In fact, job size is not the only factor considered in SDSC job scheduling. Table 2 shows the configurable metrics that affect job priority.

No	Metric name	Metric multiplier	Explanation
1	resource_weight	250.0	Increase/decrease importance based on number or CPU used
2	expansion_factor_weight	1.0	(wall_+queue)/ queue
3	system_queue_time_weight	0.020	Multiplier to age job based on system queue time
4	submit_time_weight	0.0000001	Multiplier to age job based on duration since submit time
5	local_user_weight	0.0	To lower one's priority
6	local_admin_weight	0.01175	Priority boost. To give big allocation users a high priority

7	wall_time_weight	0.0	Increase/decrease importance based on wall clock time used.
8	qos_priority_weight	400.0	Individual QOS priority boost
9	qos_target_expansion_factor_weight	1.0	Priority boost that kicks in when the job age is closer to QOS target queue time
10	qos_target_queue_wait_time_weight	1.0	Priority boost that kicks in when the job age is close to target queue time

Table 2. Catalina metric multiplier along with DS values

There are a total of 10 metric priorities; each is calculated by multiplying a metric_value by metric_multiplier.

The metric_value is derived from job property. For instance, the metric value for resource_weight is derived from the number of requested CPUs. If a job requires 32 CPU, then the resource_weight metric priority = 32 * 250 = 8000. The metric value for submit_time_weight is the number of seconds that the job waited in the queue. The sum of all priorities is the job_priority. Job_priority is an integer. Mathematically we calculate job_priority with equation (1.1).

$$job_priority = \sum (metric_value * metric_multiplier)$$

(1.1)

After the job_priority is calculated for each job, the jobs are scheduled in descending order. This way DS achieves better throughput for larger jobs while still maintaining a low expansion factor for important jobs. Furthermore allowing small jobs to run in the backfill window mitigates starvation.

In production parallel job scheduling, we want to improve the expansion factor (xfactor). Expansion factor is the ratio of the total time the job exists (the sum of runtime and time waiting in the queue) over its runtime. Hence, the ideal expansion factor is 1 where a job doesn't spend any time in the queue.

To this end, we introduce the priority-weighted utilization and priority-weighted expansion factors. Unlike its non-priority-weighted counterpart, priority-weighted expansion factor (priority_weighted_xfactor) takes individual job_priority into account. This is described mathematically as (1.2). job_priority is the individual job_priority value we obtained from equation (1.1). wait_time is the number of seconds that the job has been waiting in the queue. run_time is the number of seconds that we project the job to run. run_time is also known as wall clock time. Please note that priority_weighted_xfactor is associated with each job. To get the complete picture, or ag-

aggregate priority_weighted_xfactor, one needs to sum all of the priority_weighted_xfactor for each job and divide it by the number of jobs. Unlike the traditional xfactor, the ideal value for aggregate priority_weighted_xfactor is not 1. Our goal is to have as small an aggregate value as possible.

$$priority_weighted_xfactor = job_priority * \frac{(wait_time + run_time)}{run_time}$$

(1.2)

Utilization is a value measured by a percentage that represents the ratio of the number of seconds when a job is active in a CPU to the number of seconds in sampling period. 100% is the ideal utilization value. For instance, if we have a three-node cluster and node1 is active 50% of the time, node2 is active 35% of the time, and node3 is active 20% of the time, we have a utilization of $(50+35+20)/(100+100+100) = 35/100 = 35\%$.

However, to reiterate the priority calculation, we introduce another metric called priority_weighted_utilization. Each job has a priority_weighted_utilization value calculated according to equation (1.3). Recall that we achieved individual job_priority from equation 1.1. Multiplying each job_priority with a global utilization value gives us an individual job priority_weighted_utilization. aggregate_priority_weighted_utilization is achieved by averaging all jobs' priority_weighted_utilization. Our goal is to increase this value.

$$priority_weighted_utilization = job_priority * utilization$$

(1.3)

Obviously job priority plays an integral role in these calculations. The aggregate priority_weighted_xfactor measures the overall expansion factor of the system. Thus, if Catalina preempts a job with lower priority in favor of one with higher priority, the aggregate priority_weighted_xfactor will decrease. This is true because preempting lower priority jobs will not increase aggregate priority_weighted_xfactor as much due to the small multiplier. Running high priority jobs with small xfactor guarantees low aggregate priority_weighted_xfactor.

The aggregate priority_weighted_utilization measures the overall system utilization based on the preset priority calculation. Aggregate utilization will also improve if we follow Catalina preemption rules by preempting lower priority jobs. This is true because priority_weighted_utilization increases proportionally with job_priority (see equation 1.3).

Hence we have shown that Catalina's preempting rules allow SDSC supercomputers to improve their aggregate priority_weighted_utilization and their aggregate priority_weighted_xfactor simultaneously.

5. Benchmark

We ran a simple benchmark to compare important metrics with and without job preemption. We used actual job data from DataStar system in March 2007. We selected the first 200 jobs from the list and submitted them into Catalina in simulation mode. We then submitted another 100 jobs into the queue and measured expansion-factor and queue wait time. In the second run, we marked the first 200 jobs as preemptible and the next set of jobs to be preempting, only if their size is larger than 64 nodes, else they are preemptible, too.

To enable preemption, we increased the priority of the preempting jobs, to trigger preemption in accordance with Catalina's second rule that says "A job may preempt a running job only if the preempting job has a higher priority". After the system priority update, this incoming job preempted other running jobs of lower priority. In fact, we multiplied the priority of the preempting job by two to simulate the scenario where a job is very important and deserves preempting access. Without preemption, this job would have waited for 7 hours and 43 minutes.

Next, we calculated the average expansion factors and queue wait time weighted by job priority or node seconds. To calculate priority weight, take the ratio of the job's priority over the sum of all job's priority. To calculate node second weight, take the ratio of node-second product over the sum of all jobs' node-second product. We weighted the numbers because not all jobs are considered equally important. We provide two weighted values for comparison. Smaller numbers are better.

Avg queue wait (seconds)	w/out preemption	w/ preemption
Non-weighted	90055.31	91503.22
Priority weighted	59413.56	46489.23
Node second weighted	102979.93	99626.32

Table 3. Average queue wait times

Avg. Expansion factor	w/out preemption	w/preemption
Non-weighted	8.57	8.24
Priority weighted	11.36	4.43
Node second weighted	3.95	3.39

Table 4. Average expansion factors

Utilization (%)	w/out preemption	w/preemption
Non-weighted	88.36	88.54

Table 5. Utilizations over three days timeframe.

Looking at the tables, considering both weights, Catalina with preemption yielded better results. This was largely due to the fact that the preempting job that carries much weight was scheduled first and therefore suffered no delay, therefore the average queue wait time was lower. Expansion factor also depends on queue wait time according to the formula, therefore Catalina with preemption yielded better results here too.

Comparing the two weights together, we found that priority weight is closer to reality than node-second weight because high priority jobs need not be large. A medium job to simulate an imminent earthquake is more important than a full machine job, to put them in perspective.

We didn't find any significant change in terms of utilization of the machine with and without preemption, at least by looking at the time window of three days ahead.

6. Limitations of Preemption

Of course preemption also has its cost. In this paper we will only look into SDSC's DataStar. DataStar is a 16TF IBM POWER4 supercomputer running IBM AIX 5.2L and LoadLeveler 3.3.1.2. Catalina is the batch scheduler for DataStar.

Kettimuthu[1] argued that limiting factor for preemption is the time to write to disk. We believe that there are additional factors that could limit preemption effectiveness. During runtime, a job will exclusively occupy resources. Some resources that LoadLeveler jobs use include processor, scheduling slots, real memory, consumable CPU, consumable memory, and communication switch [2]. When this job is preempted, all of the resources have to be freed in order for that node to reach its original idle state (when the job wasn't running). In LoadLeveler, however this can't be achieved without some artifacts. Any preempted job in backfill or gang scheduling mode would still hold on to its consumable virtual memory and floating resources [2]. We argue that the inability to restore the node to its original state during preemption sets our theoretical peak of how much preemption is possible with this class of supercomputers.

Catalina only allows one level of preemption, i.e. a preempted job cannot be preempted by another job. This does limit our flexibility of achieving the desired average

priority_weighted_xfactor, but we believe that this is the practical way to do it. Additional jobs suspended on the same set of nodes may exhaust swap space.

7. Catalina on SDSC's Supercomputers

Catalina has been running in SDSC's production resources since 1999. Catalina first ran on Blue Horizon, our first Teraflop machine. Then we ran Catalina on our TeraGrid Linux [10] machine. Afterwards we ran it on DataStar. The code is functional and stable for production supercomputers. We favor the Catalina scheduler because of its flexibility, performance, extensibility, and customizability.

Catalina is tailored to our needs and the needs of our user base. We can tweak Catalina settings on-the-fly and experience the effect in the next scheduling interval.

System	Arch/OS	Re-source Manager	Size (TF)	Year
Blue Horizon	POWER3/AIX 5L	LoadLeveler	1.7	1999-2004
TeraGrid Linux	Itanium IA64/SuSe Linux	Torque (OpenPBS)	4.0	2002-present
DataStar	POWER4/AIX 5L	LoadLeveler	16.0	2003-present

Table 6. Production systems with Catalina scheduler.

The Catalina preemption code is currently running on both the TeraGrid Linux and DataStar machines.

8. Future Work

Our next step is to continue our practical evaluation of Catalina's implementation of preemption by quantitatively analyzing data from DataStar's production job log. The log will serve as our model to gain insight into scheduling effectiveness as measured by achieved aggregate priority_weighted_utilization aggregate and priority_weighted_xfactor.

9. Conclusion

By adding preemption functionality to our local scheduler, Catalina, we are able to offer new services to the users of our production supercomputers. With an explanation of the priority calculations on one of our supercomputers, we explore how priorities are calculated and how preemption affects utilization in a positive way. We also explored some of the limitations of preemption. With this new capability we plan to monitor and analyze the metrics closely to adjust our priority calculations to make our machines more highly utilized.

10. References

- [1] R. Kattimuthu, et.al, "An Evaluation of Preemption Strategies for Parallel Job Scheduling".
- [2] G. Sabin, et.al, "Scheduling of Parallel Jobs in a Heterogeneous Multi-site Environment", Proc of 9th workshop on Job Scheduling Strategies for Parallel Processing, 2003.
- [3] A. Youssef, et.al, "A Comparative Study of Two Generic Resource Allocation Models", Proc of 3rd Int'l Conf. on Computer Science & Informatics, 1997, pp. 47-50.
- [4] Catalina scheduler. www.sdsc.edu/catalina.
- [5] IBM LoadLeveler. <http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp>.
- [6] Portable Batch System (PBS). <http://www.openpbs.org>.
- [7] Global Grid Forum, Grid Resource Allocation agreement Protocol(GRAAP-WG). <https://forge.gridforum.org/projects/graap-wg>.
- [8] D. Jackson, et.al, "Core Algorithms of the Maui Scheduler.", Proc. 7th Workshop on Job Scheduling Strategies for Parallel Processing, 2001.
- [9] K. Yoshimoto, et.al, "Co-scheduling with User-Settable Reservations", Proc 11th Workshop on Job Scheduling Strategies for Parallel Processing, 2005.
- [10] <http://www.teragrid.org>