

# **SANDIA REPORT**

SAND2008-1310

Unlimited Release

Printed February 2008

## **Parallel Job Scheduling Policies to Improve Fairness: A Case Study**

Vitus J. Leung, Gerald Sabin, and P. Sadayappan

Prepared by  
Sandia National Laboratories  
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

**NOTICE:** This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from  
U.S. Department of Energy  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831

Telephone: (865) 576-8401  
Facsimile: (865) 576-5728  
E-Mail: [reports@adonis.osti.gov](mailto:reports@adonis.osti.gov)  
Online ordering: <http://www.osti.gov/bridge>

Available to the public from  
U.S. Department of Commerce  
National Technical Information Service  
5285 Port Royal Rd.  
Springfield, VA 22161

Telephone: (800) 553-6847  
Facsimile: (703) 605-6900  
E-Mail: [orders@ntis.fedworld.gov](mailto:orders@ntis.fedworld.gov)  
Online order: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



SAND2008-1310  
Unlimited Release  
Printed February 2008

## Parallel Job Scheduling Policies to Improve Fairness: A Case Study

Vitus J. Leung  
Computer Science & Informatics Department  
Sandia National Laboratories  
P.O. Box 5800  
Albuquerque, NM 87185-1318  
Gerald Sabin and P. Sadayappan  
The Ohio State University  
Columbus, OH 43210

### **Abstract**

Balancing fairness, user performance, and system performance is a critical concern when developing and installing parallel schedulers. Sandia uses a customized scheduler to manage many of their parallel machines. A primary function of the scheduler is to ensure that the machines have good utilization and that users are treated in a “fair” manner. A separate compute process allocator (CPA) ensures that the jobs on the machines are not too fragmented in order to maximize throughput.

Until recently, there has been no established technique to measure the fairness of parallel job schedulers. This paper introduces a “hybrid” fairness metric that is similar to recently proposed metrics. The metric uses the Sandia version of a “fairshare” queuing priority as the basis for fairness. The hybrid fairness metric is used to evaluate a Sandia workload. Using these results, multiple scheduling strategies are introduced to improve performance while satisfying user and system performance constraints.

## **Acknowledgments**

Thanks to Jeanette Johnston for the discussions regarding the previous policy and possible improvements. Thanks to Jon Stearley for going out of his way to discuss “fairness” and for getting the raw Cplant logs.

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>Sandia Environment</b>	<b>9</b>
2.1	Scheduler . . . . .	9
2.2	Workload . . . . .	10
<b>3</b>	<b>Simulation Environment</b>	<b>14</b>
3.1	Simulator . . . . .	14
3.2	Standard Metrics . . . . .	14
3.2.1	User Metrics . . . . .	14
3.2.2	System Metrics . . . . .	15
<b>4</b>	<b>Fairness Metrics for Parallel Job Scheduling</b>	<b>15</b>
4.1	A Hybrid “Fairshare” Metric . . . . .	16
<b>5</b>	<b>Fairness Directed Policies</b>	<b>17</b>
5.1	Maximum Runtime Limits . . . . .	18
5.2	Limit Entrance to the Starvation Queue . . . . .	18
5.3	Conservative Backfilling . . . . .	18
5.4	Conservative Backfilling with Dynamic Reservations . . . . .	19
5.5	Scheduling Policies Presented . . . . .	19
<b>6</b>	<b>Results</b>	<b>20</b>
6.1	Minor Changes . . . . .	20
6.2	Conservative Backfilling Results . . . . .	25
<b>7</b>	<b>Conclusions</b>	<b>27</b>

## List of Figures

1	Simple FCFS Backfill Example . . . . .	7
2	FCFS Backfill Example . . . . .	8
3	Load of Ross Workload . . . . .	10
4	Load of Ross Workload . . . . .	11
5	Load of Ross Workload . . . . .	13
6	Load of Ross Workload . . . . .	13
7	Load of Ross Workload . . . . .	14
8	Percent of Missed Jobs (Minor Changes) . . . . .	20
9	Average Miss Time (Minor Changes) . . . . .	21

10	Average Miss Time (Width Categories, Minor Changes) . . . . .	21
11	Average Turnaround Time (Minor Changes) . . . . .	22
12	Average Turnaround Time (Width Categories, Minor Changes) . . . . .	23
13	Loss Of Capacity (Minor Changes) . . . . .	23
14	Percent of Missed Jobs . . . . .	24
15	Average Miss Time . . . . .	24
16	Average Miss Time (Width Categories, Conservative) . . . . .	25
17	Average Turnaround Time . . . . .	26
18	Average Turnaround Time (Width Categories, Conservative) . . . . .	26
19	Loss of Capacity . . . . .	27

## List of Tables

1	Categorywise Job Count . . . . .	12
2	Categorywise Proc-Hr . . . . .	12

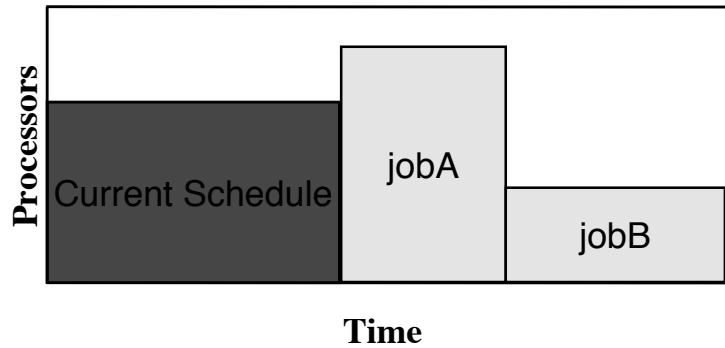


Figure 1: An example of a simple FCFS schedule without backfilling

## 1 Introduction

Clusters and other supercomputers often use parallel job schedulers [2, 3, 4] to dynamically determine the jobs execution order and, in some cases, which nodes to allocate to each job. Users submit jobs to a scheduler (e.g., qsub), giving information such as expected runtime and the required node allocation size. The scheduler is responsible for determining when to start each job. There has been much research evaluating various non-preemptive, space shared job scheduling strategies [10].

The problem can be viewed in terms of a 2D chart with time along one axis and the number of processors along the other axis. Each job can be thought of as a rectangle whose length is the user estimated run time and width is the number of compute nodes required. The scheduler’s role is to “pack” sets of jobs into the 2D schedule. Users can submit new jobs to the system, that need to be incorporated into the current schedule. Therefore, the schedule must be able to handle dynamically arriving jobs of various sizes. Schedulers inherently use a queue to store jobs that have arrived but have not been launched or started. The generated schedules must be sensitive to both user and system needs such as: how long does it take for each user’s job to run and how well the system resources are being utilized.

The simplest way to schedule jobs at a single site is to use a strict First-Come-First-Serve (FCFS) policy. However, this approach suffers from low system utilization [16]. A strict FCFS policy ensures that jobs are started in the order of arrival. Therefore, only jobs from the head of the queue can be started. A job that is not at the head of the queue must wait, even if there are currently enough resources available. For instance, in Figure 1 *jobB* can not start, even though there are enough resources available. Therefore, a strict FCFS

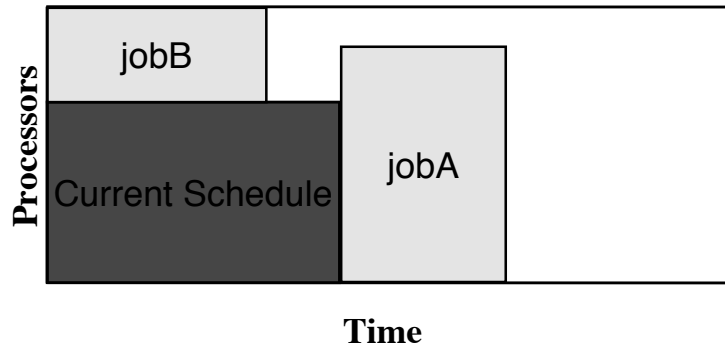


Figure 2: An example of a backfill in an FCFS backfilling schedule

policy is “fair” but leads to poor utilization and a poor average turnaround time.

Backfilling [22, 23] was proposed to help improve system utilization and has been implemented in most production schedulers [14, 28]. Backfilling is the process of starting a job that is lower in the queuing priority order before the job that is at the head of the queue. Backfilling identifies “holes” in the 2D chart and moves forward smaller jobs that fit into these holes, without delaying any jobs with future “internal” reservations. This helps improve utilization, by not allowing processors to remain idle if there is a job that fits in a *hole*, and helps to reduce average turnaround time due to the increased utilization. Figure 2 shows a similar situation to Figure 1, except *jobB* is now allowed to start due to backfilling.

We will now define what it means for a job to fit into a *hole* in the schedule. A backfilling scheduler creates internal *reservations* for some of the jobs. These reservations, as well as time blocked off for running jobs, provide a schedule in which jobs are allowed to backfill. A *hole* is an open space in this 2D chart. Backfilling allows a job that fits into this schedule to improve its internal reservation (by obtaining an earlier time slot), as long as it fits into a hole and does not violate any other reservations.

There are two common variations to backfilling - *conservative* and *aggressive* (EASY)[11, 28]. In conservative backfilling, every job is given an internal reservation when it enters the system. A smaller job is moved forward in the queue as long as it does not delay any previously queued job. In aggressive backfilling, only the job at the head of the queue has a reservation. A small job is allowed to leap forward as long as it does not delay the job at the head of the queue. *No guarantee* backfilling is a less often used variation. In no guarantee backfilling, no jobs are given reservations. This has the possibility of leading



to starvation (see below) and is therefore not often used. Many production schedulers use variations between conservative and aggressive backfilling, giving the first  $n$  jobs in the queue a reservation.

Other parallel job scheduling techniques have been designed to reduce the turnaround time for users [8, 17, 12] and increase utilization [16, 18] in a “fair” environment. Until recently, fairness was not a primary concern in much of the research literature. However, fairness has always been a primary concern when setting up a parallel job scheduler.

This fairness concern is evident in the scheduler developed and put into production at Sandia National Laboratories on various machines (e.g., CPlant [1]). The Sandia scheduler prioritizes jobs using a decaying processor-time value. This value tracks the usage of each user and decays on a regular basis. This attempts to provide users who have not recently used the machine priority over other users. The intent of this queuing priority is to provide a sense of fairness amongst the users.

The remainder of this paper is organized as follows: Section 2 examines the original Sandia scheduling policies and workload studied in this paper. Section 3 discusses the simulation methodologies used in this study. Section 4 reviews existing fairness metrics for parallel job schedulers and introduces a “hybrid” metric. Section 5 introduces a few scheduling policies designed to reduce unfairness. Section 6 examines the effects of the scheduling policies introduced in Section 5.

## 2 Sandia Environment

This study examines the CPlant/Ross machine at Sandia National Laboratories. The scheduling policy and workload logs were required to perform this case study. The scheduler policy was obtained via the CPlant website and personal communications with Jeanette Johnston. The raw workload logs (PBS and yod logs) were obtained with assistance from Jon Stearley.

### 2.1 Scheduler

The baseline scheduler in use (at the time the study was completed) on the CPlant machine was a no guarantee backfill variant. The queuing policy was based on a “fairshare” queuing priority aimed at providing a level of user fairness. The “fairshare” queuing order was determined by a historical sum of processor-seconds used that decayed every 24 hours. This provided priority to users who had not recently used the machine. There were no internal reservations. At each scheduling event (job completion and job arrival), the queue was processed in fairshare priority order; if there were sufficient nodes, a job was started (i.e., no guarantee backfilling). This has been shown to negatively affect wide jobs, as it is unlikely that enough nodes will be free for a wide job to start, as lower priority, narrower jobs will be allowed to start ahead of it.

To prevent wide jobs from starving, a secondary “starvation” queue was used. The starvation queue used an FCFS priority order, rather than “fairshare”. The head of the star-

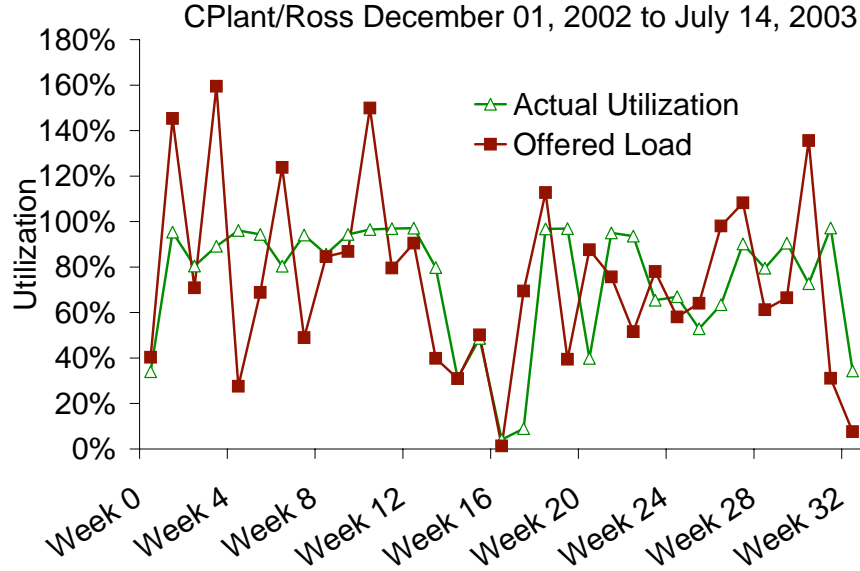


Figure 3: The offered load and actual utilization of the CPlant/Ross workload between December 1st 2002 and July 14th 2003

vation queue received an internal reservation (i.e., aggressive backfilling), and thus progress was guaranteed.

## 2.2 Workload

Workload logs from the CPlant system from December 01, 2002 to July 14, 2003 were collected and processed for use in this study. The trace was converted to the Standard Workload Format (SWF V2) from multiple system logs (PBS and the job launcher, yod). Effort was taken to track the user id, group id, start time, completion time, submit/queue time, wall clock limit (i.e., user estimated runtime) and nodes requested. The timing and node information are required to characterize the shape of a job. The user id's are required to compute the "fairshare" value for the Sandia scheduling policy. User and group id's were replaced sequentially (e.g., the first user is given an id of 1) to remove the actual user and group id's for public release. A superset of the traces will be released via the workload archive [9] soon.

The trace contains 13614 jobs over the 7.5 months (231 days). The trace contains periods of very high utilization (over 90%), see Figure 3. The offered load shows the amount of queued workload over time, while the utilization shows the actual achieved utilization. The CPlant workload contains many weeks where the offered load is much greater than 100%, implying that not enough resources are available to complete the work given to the system in that time period. High load weeks are often followed by weeks where the load is much lower. These cyclic low load periods are likely due to the users submitting

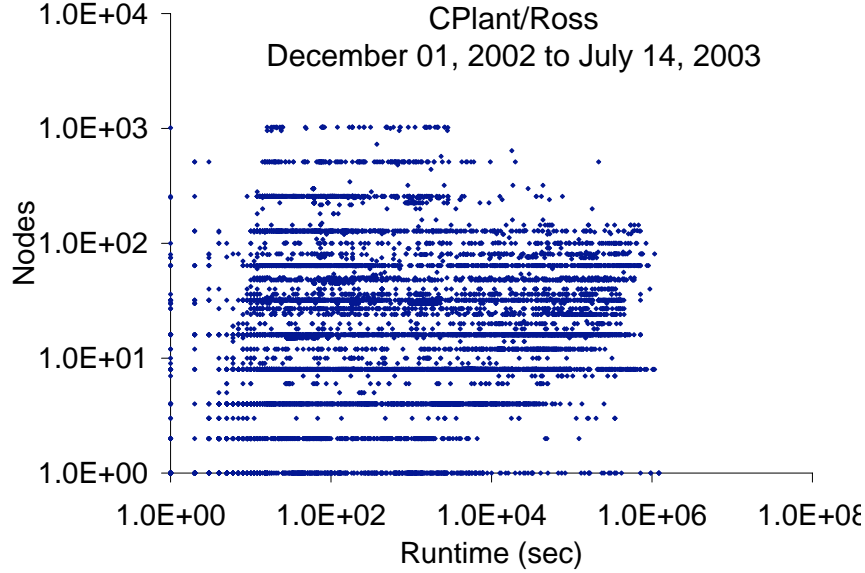


Figure 4: The runtime and node usage for the CPlant/Ross workload between December 1st 2002 and July 14th 2003

fewer jobs due to the extremely high queue lengths and wait times.

Figure 4 plots the submitted jobs. Many users choose “standard” node allocations that are powers of two or squares, as seen in other workloads [9, 13]. Table 1 shows that most of the jobs are short; few jobs use 2-4 nodes, and few jobs use more than 128 nodes. However, there are quite a few very long jobs in the workload. Table 2 shows the same data as Table 1, but in total processor-hours instead of just number of jobs in each category. This shows that even though there are fewer longer jobs than short jobs, the wide and long jobs represent a significant portion of the workload.

Figure 5 plots user estimates vs. actual runtimes for each job. The custom PBS scheduler kills jobs after the user supplied wall clock limit (WCL) is reached. However, if no other job requires the processors, the job is allowed to continue running until the processors are needed. This results in a few jobs having longer runtimes than estimated. The process of killing jobs and the effect job placement has on runtimes [21] lead to many users providing user estimates that are much longer than the expected runtime. The intentional over estimations, combined with unknown system and networking contention and jobs that abort unexpectedly explain much of the overestimation seen. Attempts to reduce networking contention are documented in [5], [6], [7], [20], and [21]. Figures 6 and 7 show any correlation between runtime and nodes, respectively, and the overestimation factor.

Table 1: Number of jobs in each length/width category

	<b>0-15 mins</b>	<b>15-60 mins</b>	<b>1-4 hrs</b>	<b>4-8 hrs</b>	<b>8-16 hrs</b>	<b>16-24 hrs</b>	<b>1-2 days</b>	<b>2+ days</b>
<b>1 node</b>	681	141	44	7	7	3	6	16
<b>2 nodes</b>	458	80	8	0	2	0	1	0
<b>3-4 nodes</b>	672	440	273	55	26	3	5	5
<b>5-8 nodes</b>	832	238	700	155	142	90	76	91
<b>9-16 nodes</b>	1032	131	347	206	260	141	205	160
<b>17-32 nodes</b>	917	608	113	72	67	53	116	160
<b>33-64 nodes</b>	879	130	134	70	79	48	130	178
<b>65-128 nodes</b>	494	72	78	31	49	24	53	76
<b>129-256 nodes</b>	447	127	9	5	12	1	3	10
<b>257-512 nodes</b>	147	24	6	3	1	0	0	1
<b>513+ nodes</b>	51	18	1	0	0	0	0	0

Table 2: Processor-hours in each length/width category

	<b>0-15 mins</b>	<b>15-60 mins</b>	<b>1-4 hrs</b>	<b>4-8 hrs</b>	<b>8-16 hrs</b>	<b>16-24 hrs</b>	<b>1-2 days</b>	<b>2+ days</b>
<b>1 node</b>	14	61	76	42	70	62	259	2883
<b>2 nodes</b>	32	70	21	0	53	0	68	0
<b>3-4 nodes</b>	103	1197	2210	1272	1030	213	614	1310
<b>5-8 nodes</b>	281	1101	10263	6582	12107	14118	18287	92549
<b>9-16 nodes</b>	522	1102	12522	18175	45859	42072	105884	207496
<b>17-32 nodes</b>	968	6870	6630	11008	22031	28232	109166	363944
<b>33-64 nodes</b>	1775	2895	15252	20429	48457	48493	251748	986649
<b>65-128 nodes</b>	1876	4149	19125	17333	53098	48296	179321	796517
<b>129-256 nodes</b>	3273	12395	4219	4322	27041	5451	19030	183949
<b>257-512 nodes</b>	3719	4723	5027	6850	3888	0	0	30761
<b>513+ nodes</b>	2692	9503	0	3183	0	0	0	0

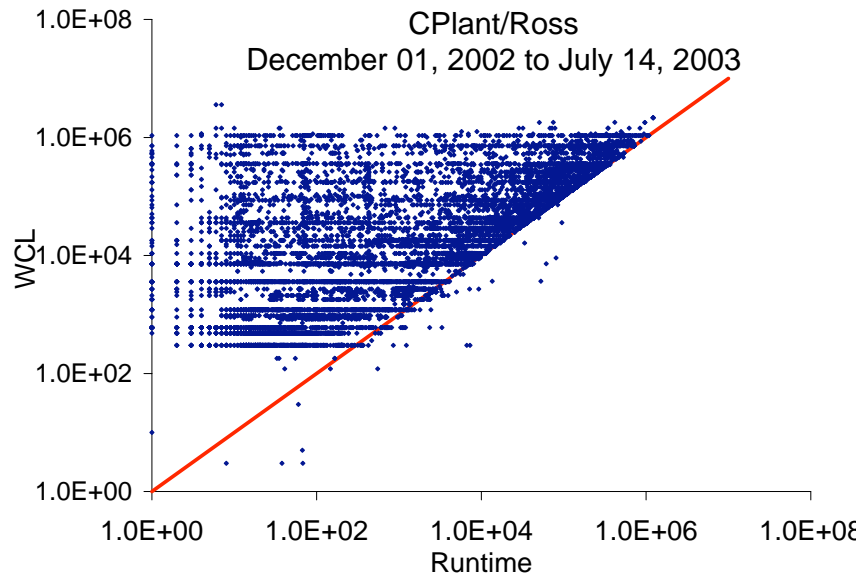


Figure 5: User estimates for the CPlant/Ross workload between December 1st 2002 and July 14th 2003

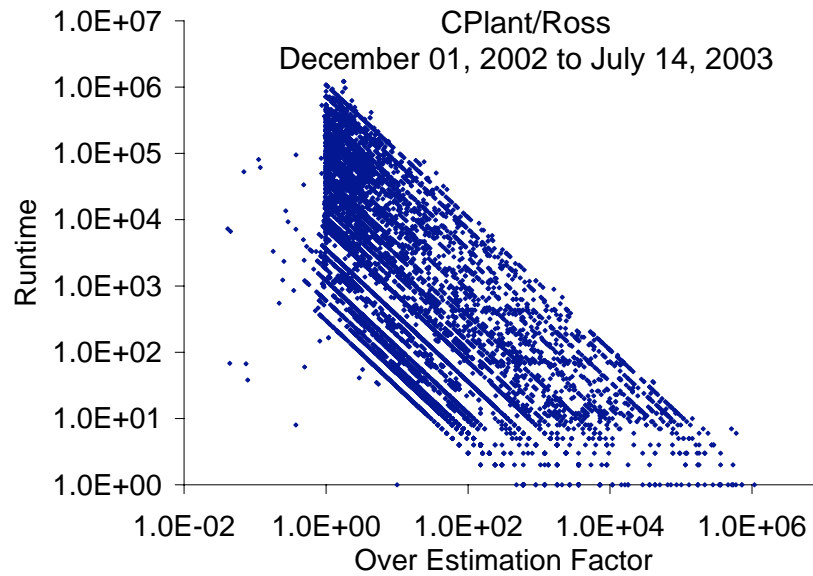


Figure 6: The overestimation factor reduces for longer jobs in the CPlant/Ross workload between December 1st 2002 and July 14th 2003

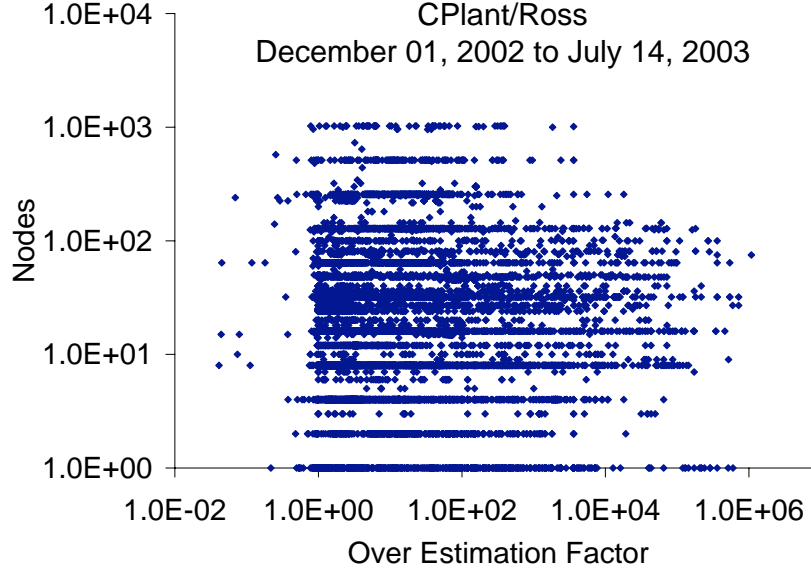


Figure 7: The overestimation factor appears unrelated to the node selection in the CPlant/Ross workload between December 1st 2002 and July 14th 2003

### 3 Simulation Environment

#### 3.1 Simulator

A locally developed event based simulator was used to simulate various scheduling policies using the CPlant workload log. The simulator can simulate multiple queuing orders and reservation depths. The necessarily modification were made to simulate any of the scheduling algorithms presented. The scheduler takes as input a trace file in the Standard Workload Format V2 [9].

#### 3.2 Standard Metrics

Parallel job scheduling metrics can be divided into two major categories: user and system metrics. User metrics are designed to measure the performance of a particular schedule from a users point of view. System metrics measure the performance from a “system” or administrative point of view.

##### 3.2.1 User Metrics

Common user metrics include wait time, turnaround time, and slowdown. Waittime measures the time between  $job_i$ 's arrival and  $job_i$ 's start time. Turnaround time measures the time between  $job_i$ 's arrival time and its completion time.

$$AverageTurnaroundTime = \frac{\sum_{j \in jobs} j.completetion\_time - j.arrival\_time}{\sum_{j \in jobs} 1} \quad (1)$$

### 3.2.2 System Metrics

Utilization is the most common system metric. However, in simulation based studies, utilization is a poor measure of performance. In simulation studies, utilization simply is an indirect measure of makespan, as the workload of all schedulers is a constant.

$$Utilization = \frac{\sum_{i=1}^n job_i.used\_processors * job_i.runtime}{Makespan * SystemSize}, \quad (2)$$

where

$$Makespan = MaxCompletetionTime - MinStartTime. \quad (3)$$

Loss of Capacity (LOC) (see Equation 4) is often used in lieu of utilization. LOC measures the fraction of the processor cycles that were left idle when jobs were in the queue. LOC exists due to the non-work conserving nature of the job schedulers; a work-conserving schedule will, by definition, have a LOC of 0. LOC is a good metric to measure the system performance of parallel backfill scheduling simulations. The metric measures the extent to which the schedule is “packed”. A low LOC implies that the unused cycles are not due to the scheduling policy, but rather the offered workload. A high LOC implies that the scheduler is not able to pack the jobs, and it is not expected that increasing offered load will affect utilization.

$$LOC = \frac{\int_{t=0}^{max\_time} min(\sum_{q \in queuedJobs} q.nodes, SystemSize - \sum_{r \in runningJobs} r.nodes)}{Makespan * SystemSize} \quad (4)$$

## 4 Fairness Metrics for Parallel Job Scheduling

Recent work has introduced fairness metrics designed for the parallel job scheduling domain. Vasupongayya and Chiang [30] examine the use of common techniques to measure fairness. The standard deviation of the turnaround time and fairness index [15] are considered as a basis to measure fairness. These metrics assume that it is undesirable to have a high standard deviation; however, this is not the case for bursty workloads seen in parallel job scheduling. It is desirable that a job arriving in a low load condition (e.g., late evening) receive a much better turnaround time than a job arriving in heavy load (e.g., mid morning).

Srinivasan et. al [29] recognize that both an FCFS no-backfilling schedule and an FCFS conservative backfill schedule (e.g., unlimited reservations) provides a “fair” schedule when perfect user estimates are assumed. The schedule is “fair” in a social justice [19]

sense, as no job can be affected by a later arriving job. A no-backfill schedule is undesirable as the average turnaround time is very large and the utilization is very low. Therefore, the conservative backfill schedule assuming perfect estimates (CONS\_P) is assumed to be a “fair” schedule. The simulated start of each job in a scheduler under test, using inaccurate user estimates, is compared against the CONS\_P start time. The sum of these differences represents the “unfairness” of the schedule.

Sabin et. al. [25, 26, 27] have introduced multiple fairness metrics for parallel job schedulers. The first metric is based on defining a fair start time (FST), similar to the CONS\_P metric defined above. The CONS\_P metric has the apparent advantage of creating a single set of FSTs. However, while the feature allows simple comparisons of schedules, it detracts from its ability to accurately measure fairness. If a schedule has a higher utilization than the CONS\_P schedule, jobs run deliberately out of order can seem fair. Assume that two identically shaped jobs ( $job_A$  and  $job_B$ ) arrive at time  $t_a$  and  $t_b$ , with  $t_a < t_b$ . It is feasible for  $job_a$  to start after  $job_b$ , yet have both jobs start well before the CONS\_P FST, resulting in a schedule that appears fair via the CONS\_P metric. In an attempt to more accurately capture fairness, Sabin and Sadayappan attempt to directly measure the effect of later arriving jobs. The revised metrics calculates an FST for each job, by creating a schedule assuming no later jobs arrive. The start time in the new schedule represents the jobs FST. This has the advantage of directly measuring if a latter arriving job affected each job. This scheme allows “benign” backfilling, e.g., latter arriving jobs to start earlier if they do not affect any earlier job. A disadvantage of this technique is that the FST relies on the scheduling policy in place. While this eliminates the performance effects seen in the CONS\_P FST, it makes comparisons across different schedules difficult, as each job has a different FST in each schedule. The aggregate unfairness metric is calculated by summing the total unfairness (time each job misses its FST) or measuring the percentage of the load that misses its FST.

The second metric introduced by Sabin and Sadayappan [26] measures resource equality. The metric is inspired by networking and operational fairness metrics [24]. The metric measures to what extent each job was able to receive its “share” of the resources while in the system. The basis for this metric is that each job “deserves”  $1/N$  of the resources while in the system, where  $N$  is the number of “live” (running or queued) jobs. This metric does not rely on the scheduler in place (such as the FST based metric above), and thus can be used to compare schedules.

## 4.1 A Hybrid “Fairshare” Metric

This paper introduces an FST metric that falls somewhere between the CONS\_P metric and the FST metric introduced by Sabin and Sadayappan. The metric is intended to reduce the reliance on the actual scheduler under test (increasing the ability to use the metric globally, to compare traces) while not using a “gold standard” schedule that is “blessed” as an ideally fair schedule.

This FST metric is a hybrid of the two FST metrics above. The FST for each job is



determined using a list scheduler. A list scheduler keeps track of a completion time for each node. When scheduling a job, the earliest time that  $N$  nodes can be found is located (where  $N$  is the number of nodes required by the job). The completion time of each of the nodes is then updated to be the earliest start time plus the runtime of the job (i.e., the completion). There are fewer restraints than a no backfill scheduler, as jobs are not required to run in a strict no backfill order. However, it is more restrictive than a conservative backfill schedule, as “holes” can not be used.

In addition, the state of the scheduler upon job arrival is used as the starting state for each simulation. This is in contrast to the CONS\_P FST metric which compares start times to a complete conservative schedule. The metric differs from the previous Sabin and Sadayappan FST metric by using a CONS\_P policy in lieu of the actual scheduling policy under test.

In addition, the previous FST based metrics assume an FCFS scheduling order. Thus, the previous Sabin and Sadayappan FST metric attempts to measure the effect of latter arriving jobs, and the CONS\_P metric uses an FCFS conservative schedule. In many environments, FCFS is not considered a socially just schedule. Sandia uses the fairshare queuing priority because that queuing order is considered fair. Therefore, the hybrid metric used in this paper assumes that if all jobs were run in “fairshare” order, the scheduler is fair. Thus, the metric attempts to determine the effect of lower priority jobs on each job. Thus the hybrid FST is generated using a no backfill schedule using the fairshare queuing priority. The FST schedule is generated starting with the schedule in the state (i.e., running schedule, queued jobs) upon job arrival, eliminating many of the performance effects seen in the CONS\_P metric. Fairness priorities other than “fairshare” could be used to perform similar evaluations using different socially just priorities.

As in the previous FST based metrics, the average miss time of the unfair jobs is calculated as:

$$AverageMissTime = \frac{\sum_{j \in jobs} \max(0, j.start\_time - j.FST)}{\sum_{j \in jobs} 1}. \quad (5)$$

## 5 Fairness Directed Policies

The actual scheduling policy described in Section 4 is intended to provide good system utilization, good user metrics, and provide a fair environment for users. This section analyzes the current scheduling policy to provide algorithms designed to improve fairness, while minimally affecting utilization and turnaround time.

The original scheduling policy attempts to run jobs in a fair order by using the “fairshare” queuing priority. However, in order to improve performance, this priority order is not strictly adhered to. Jobs can run out of order via backfilling, which is essential in parallel job scheduling. However, the policy uses no internal reservations (until the job has been in the system for at least 24 hours) which tends to increase utilization but provides a

mechanism for unfairness. Without reservations, wide jobs will have a tendency to “starve” allowing narrower jobs an “unfair” advantage.

Further, the lack of internal reservations requires a secondary queue to prevent starvation. The starvation queue allows jobs to make progress regardless of fairness and is not sorted by the fairness policy. Therefore, the use of a starvation queue is another avenue to introduce unfairness.

## **5.1 Maximum Runtime Limits**

The first potential policy to help reduce unfairness is to reduce the maximum contiguous runtime of individual jobs. This mechanism would require jobs longer than a predefined threshold to be broken up into multiple smaller jobs. Reducing the maximum runtime is a mechanism to allow very coarse scale “preemption”, as long jobs must be submitted as several individual jobs. Breaking up very long jobs allows other jobs a chance to start after each “chunk” of the large job completes. This technique also has the potential to improve user and system metrics due to the coarse preemption being introduced.

Introducing runtime limits is a feasible policy on CPlant. Users currently checkpoint their jobs frequently. The checkpoints are currently used to help eliminate wasted cycles due to hardware failures. Therefore, creating the necessary checkpoints for maximum runtime limits would add minimal overhead. In addition, the Sandia staff have created scripts to allow users to start jobs from checkpointed runs. These scripts would ease the burden of restarting jobs from the checkpointed state.

The initial “live” Sandia CPlant scheduler does not impose any runtime limitations. Simulations are run using the original policy and with a runtime limitation of 72 hours, breaking longer jobs up into several 72 hour segments.

## **5.2 Limit Entrance to the Starvation Queue**

The starvation queue allows jobs the opportunity to obtain an internal system reservation after 24 hours, and the job can start regardless of whether it is “fair” to start the job. To help improve fairness, jobs from “heavy” users can be temporarily restricted from entering the starvation queue.

This technique has the advantage of being a “simple” change that will have minimal impact on users work flow and standard user and system metrics.

## **5.3 Conservative Backfilling**

Conservative backfilling gives every job an internal temporary reservation when it enters the system. In conservative backfilling, each job attempts to find a better reservation during each scheduling event. The jobs do not relinquish their current reservations unless better reservations are found. Therefore, when each job arrives, an upper bound on the wait time is established; this eliminates the need for a “starvation queue”.

The queue is still processed in “fairshare” order during each scheduling event, giving higher priority jobs the opportunity to find a better reservation before lower priority jobs. However, each job receives its initial reservation as it arrives in the system. This tends to introduce an FCFS feel to the schedule and reduce the effectiveness of queuing policies. However, the queue order is still very important due to inaccurate user estimates. Inaccurate user estimates (seen in Section 2) allow jobs to attempt to backfill. The “fairshare” queue priority allows “deserving” jobs to attempt to improve their reservations first.

## 5.4 Conservative Backfilling with Dynamic Reservations

Dynamic reservations helps to remove the “FCFS feel” from conservative backfilling. Initial reservations are no longer upper bounds on the waittime. At each scheduling event, all reservations are removed and a schedule is created in fairshare priority order. A potential issues with any conservative scheme is reduced utilization. It is important to ensure that utilization is not adversely affected.

Both this scheme and the current “no reservation” scheme provide no hard internal guarantees upon job arrival. However, the dynamic backfilling scheme prevents “fair” jobs from starving. This removes the need for a “starvation queue”.

## 5.5 Scheduling Policies Presented

The original scheduler is a no-reservation backfill scheduler with a custom “fairshare” queue order. A job is moved to the “starvation queue” 24 hours after submission (cplant24.nomax.all). The following modified scheduling policies were examined:

1. the original CPlant scheduler except jobs are not considered for the starvation queue for 72 hours, instead of 24 hours (cplant72.nomax.all);
2. the original CPlant scheduling policy except “heavy”/”unfair” users are not allowed to enter the starving queue (cplant24.nomax.fair);
3. introduce a 72 hour maximum runtime and use the original CPlant scheduling policy (cplant24.72max.all);
4. use all three of the above modifications: 72 hour maximum runtime, “unfair” users cannot enter the starvation queue, and 72 hours until jobs are considered for the starvation queue (cplant72.72max.fair);
5. a conservative backfilling scheduler with the fairshare queuing priority (cons.nomax);
6. a conservative backfilling scheduler with the fairshare queuing priority and introduce 72 hour runtime limits (cons.72max);
7. a conservative backfilling scheduler with dynamic reservations (consdyn.nomax);

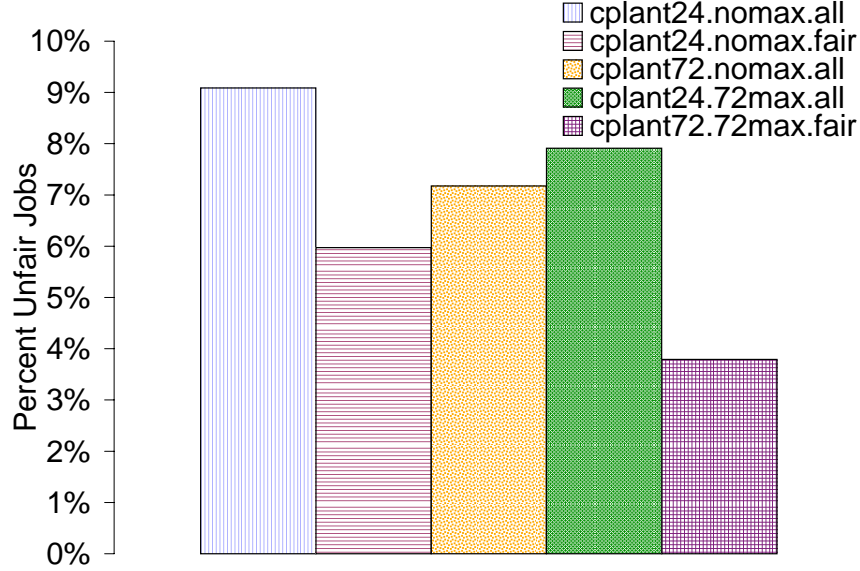


Figure 8: Percentage of jobs that missed the fair start time for the CPlant/Ross simulations

8. a conservative backfilling scheduler with dynamic reservations and 72 hour runtime limits (consdyn.72max).

## 6 Results

We group our results into two categories, minor changes and conservative backfilling.

### 6.1 Minor Changes

Increasing the time before a job is allowed in the starvation queue and/or barring “unfair” jobs from the starvation queue impose only “small” changes on the scheduler and will be mostly transparent to the users. The introduction of maximum runtimes will change the environment for the few users with very long jobs, but existing scripts will help ease the burden of the required checkpointing and restarting. These policy changes will be “easily” implemented and have a small impact on most users. In fact, it is expected that these changes will be minimally noticeable to most users who are investigating the queue status.

Figure 8 shows that all enhanced policies reduce the number of jobs that are able to start before their “fair start time”. The most improvement is seen when all three scheduling enhancements are used simultaneously. Figure 9 shows that only introducing maximum runtimes is able to reduce the average miss time. This suggests that while banning “heavy” users from the starvation queue or increasing the time until a job is allowed to starve helps to reduce the percentage of jobs that miss the fair start time, the jobs that do miss are hurt badly. Without any internal reservations, wide jobs are unlikely to get enough nodes to

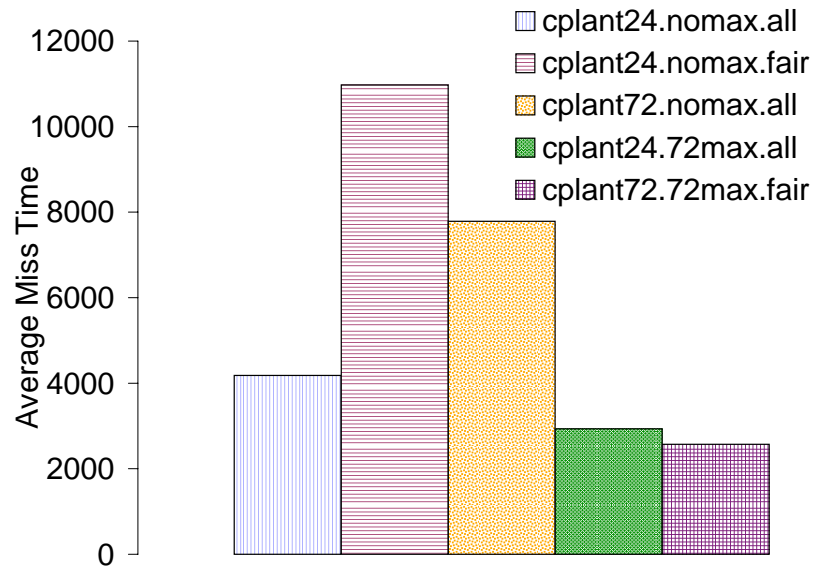


Figure 9: Average fair start miss time for initial CPlant simulations

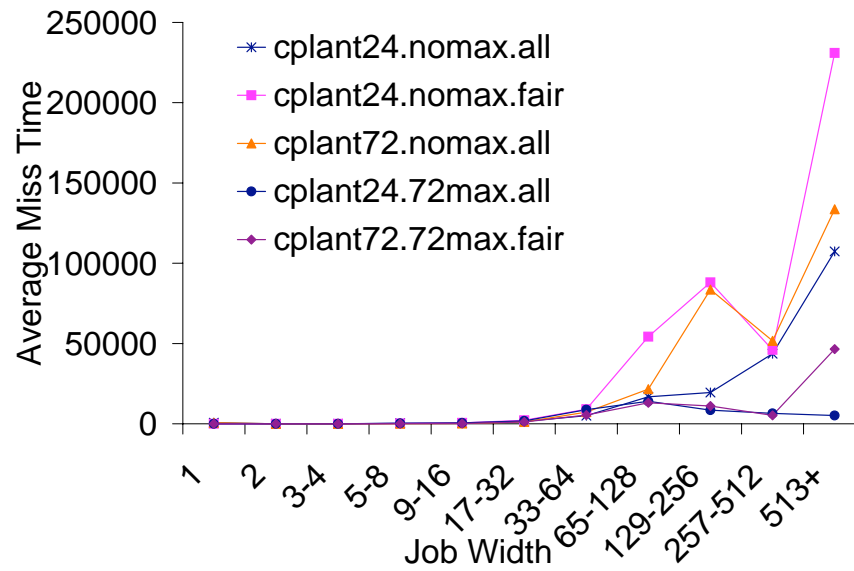


Figure 10: Average fair start miss time for initial CPlant simulations categorized by width

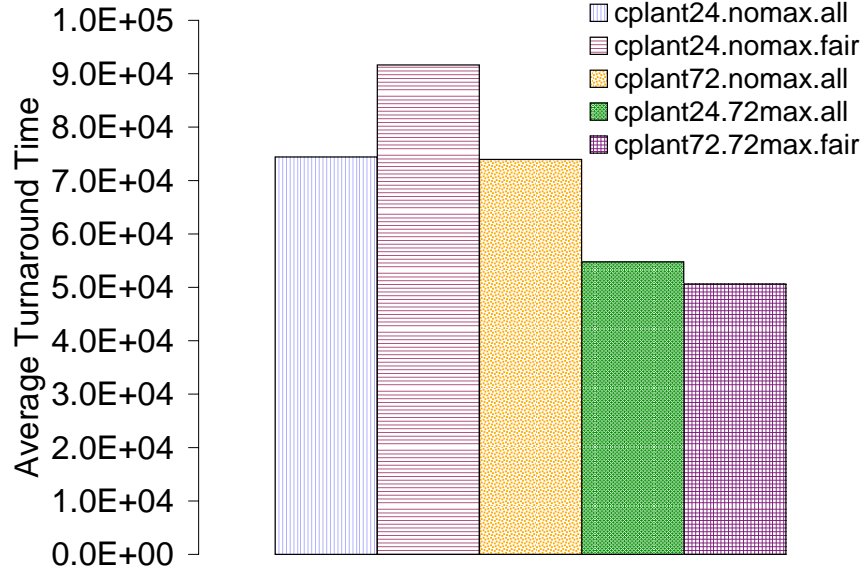


Figure 11: Average turnaround time for CPlant/Ross Simulations.

start, due to the existence of narrower jobs. These wide jobs rely on the starvation queue to start. By increasing the wait time before entering the starvation queue, the number of jobs that miss the fair start time is reduced, but the jobs that require the starvation queue to start now must wait much longer (see Figure 10).

While fairness is an important metric, it is important that the user and system metrics are not adversely affected. Figure 11 shows that the average turnaround time for the enhanced scheduling policies. The average turnaround time is improved for most of the enhanced policies. Imposing maximum runtimes on very long jobs allows for very coarse grained preemption. This allows better progress for wide jobs (see Figure 12), improving both the fairness and average turnaround time. Figure 13 shows the loss of capacity. Again, for the schedules that show an improved average miss time and an improved average turnaround time, the loss of capacity is also improved.

Introducing 72 hour maximum runtime improves the percentage of fair jobs, the average miss time, the average turnaround time, and the loss of capacity. Increasing the wait time to enter the starvation queue and disallowing “heavy” users from the starvation queue reduces the number of jobs treated unfairly, but has a negative effect on average miss time and can hurt user and system metrics. Using all three enhancements simultaneously further reduces the percent of jobs treated unfairly and the average turnaround time, but the average miss time and the loss of capacity are slightly worse than only introducing a 72 hour maximum runtime.

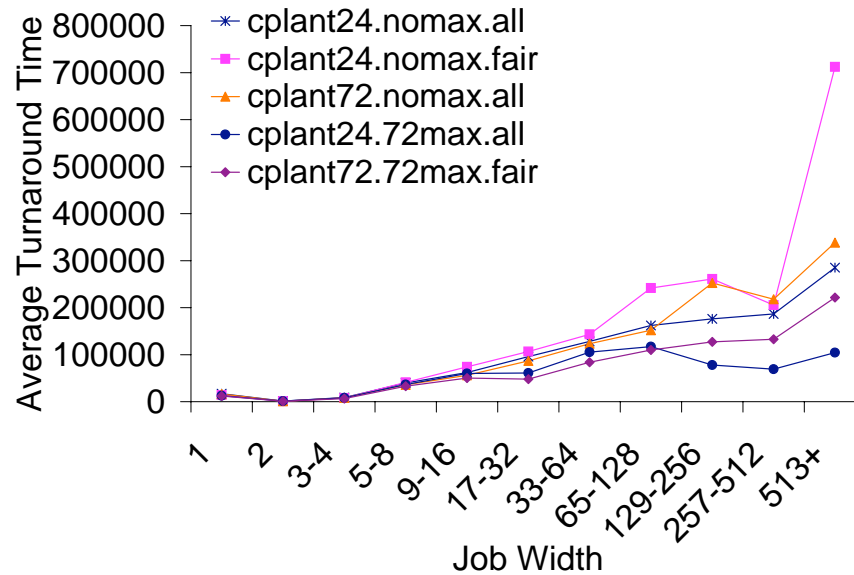


Figure 12: Average turnaround time for CPlant Simulations categorized by width

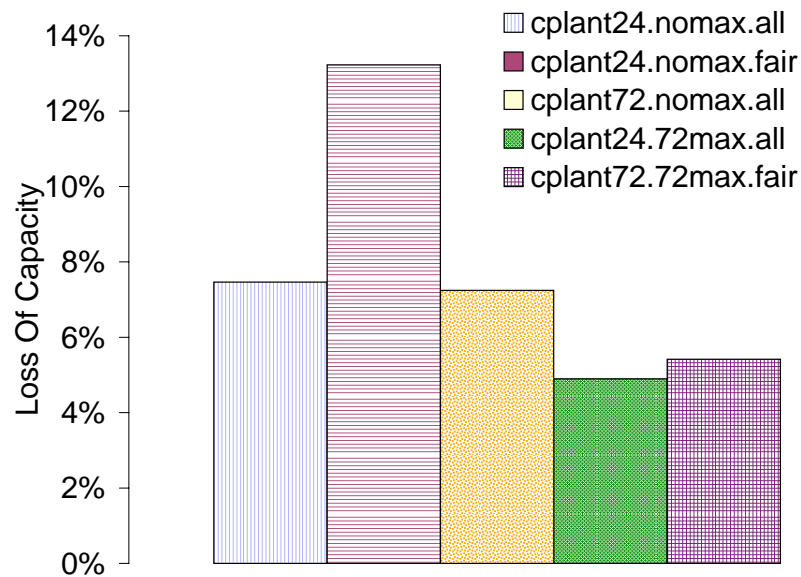


Figure 13: Loss of capacity for the CPlant/Ross simulations with “minor” changes.

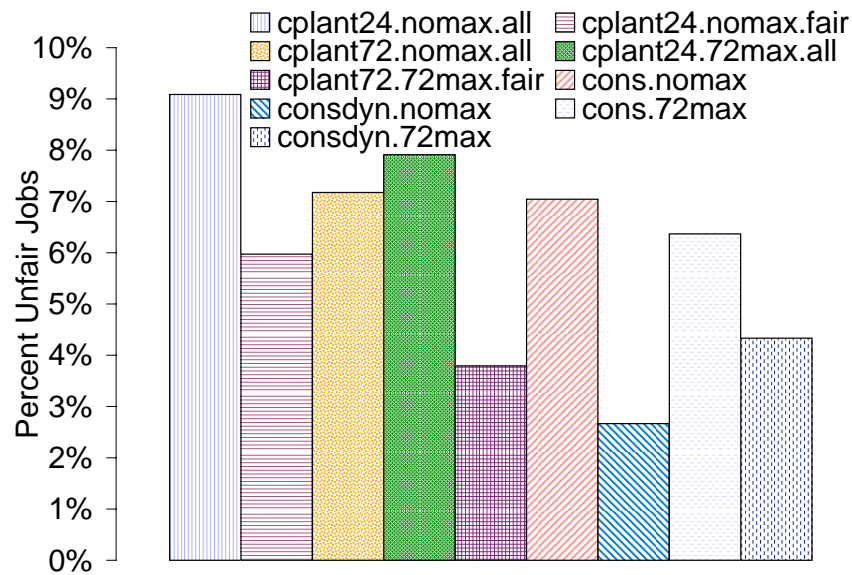


Figure 14: Percentage of jobs that missed the fair start time for all CPlant/Ross simulations

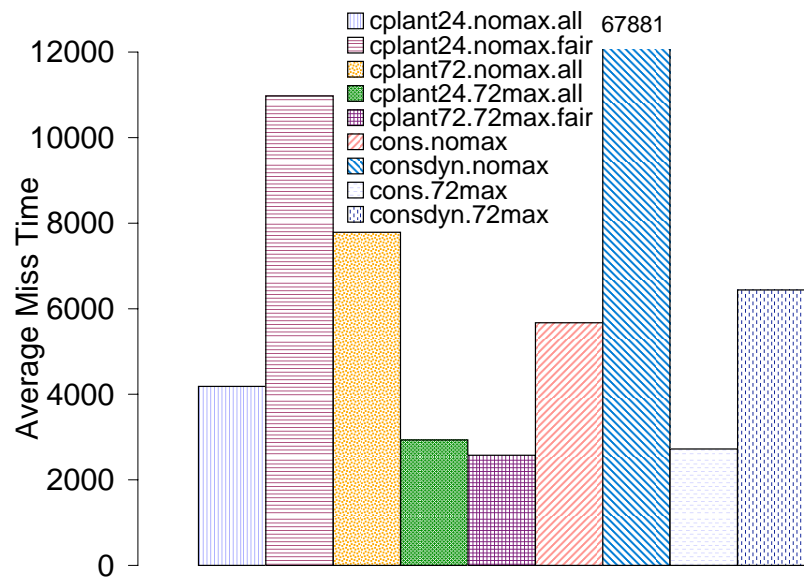


Figure 15: Average fair start miss time for all CPlant/Ross simulations



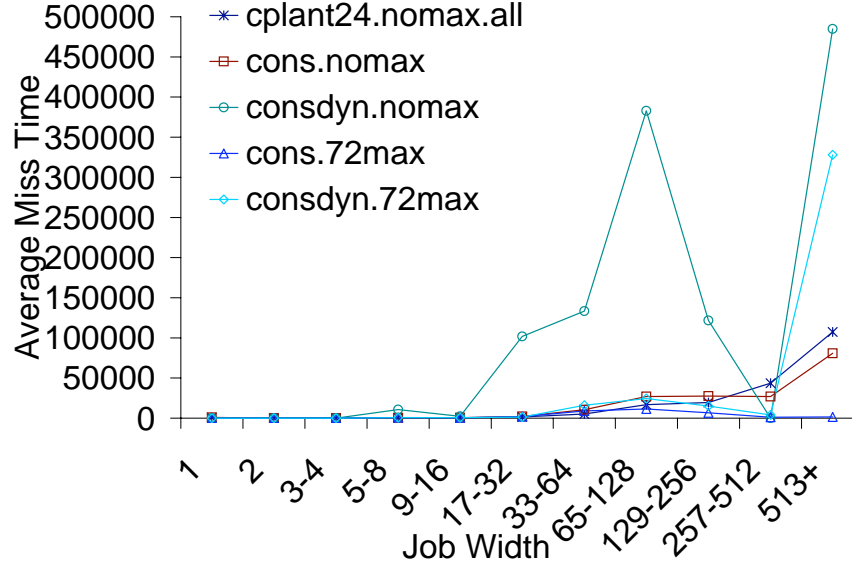


Figure 16: Average miss time for the CPlant/Ross conservative backfilling simulations categorized by width

## 6.2 Conservative Backfilling Results

Figure 14 shows the percentage of jobs that miss their fair start time. All conservative scheduling policies outperform the original policy. However, without a 72 hour runtime limitation, the conservative scheduling policies have a higher average miss time than the current policy (see Figure 15). A conservative dynamic scheduling policy has the fewest unfair jobs, but the jobs that do miss are treated very unfairly. The only policy to show a marked improvement in both percent of unfairly treated jobs and average miss time is the conservative backfilling policy with 72 hour maximum runtime limitations. In all cases, a 72 hour runtime limitations appears to be an important feature to improve system wide fairness. The conservative scheme with 72 hour limits appears to be a very competitive scheme. In addition, the conservative backfilling scheme is able to reduce the unfairness of wide jobs (see Figure 16 ), which is important as the supercomputers are purchased to efficiently run parallel code that would otherwise require a very large sequential runtime.

Figure 17 shows the average turnaround time for all policies; Figure 18 shows the average turnaround time for conservative scheduling policies categorized by width; and Figure 19 shows the lost of capacity for all policies. Conservative scheduling policies often have poor average turnaround time and utilization. However, the introduction of 72 hour job limits appears to improve the performance of the conservative schedules. The conservative schedule with 72 hour job limits has a superior average turnaround time and a lower loss of capacity than most of the other schemes. The coarse grained preemption allows for better schedule packing and a reduction in average turnaround time.

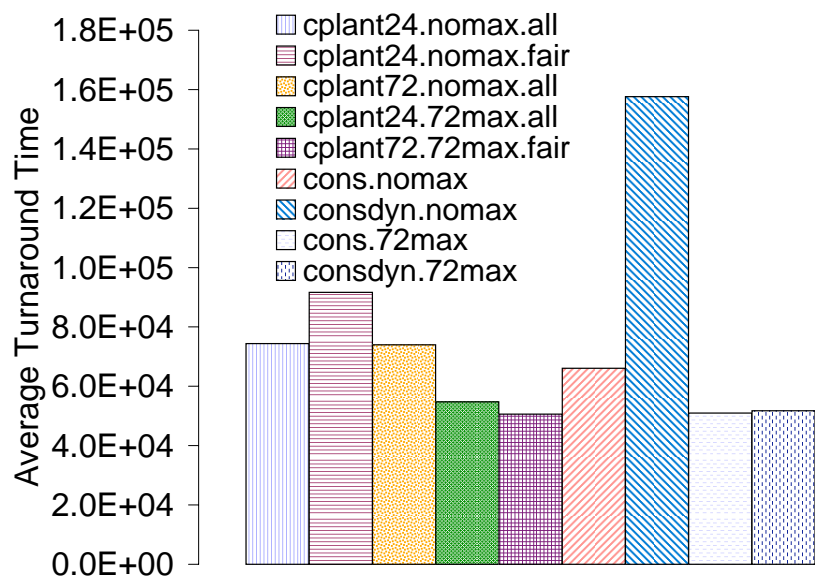


Figure 17: Average turnaround time for all CPlant/Ross simulations

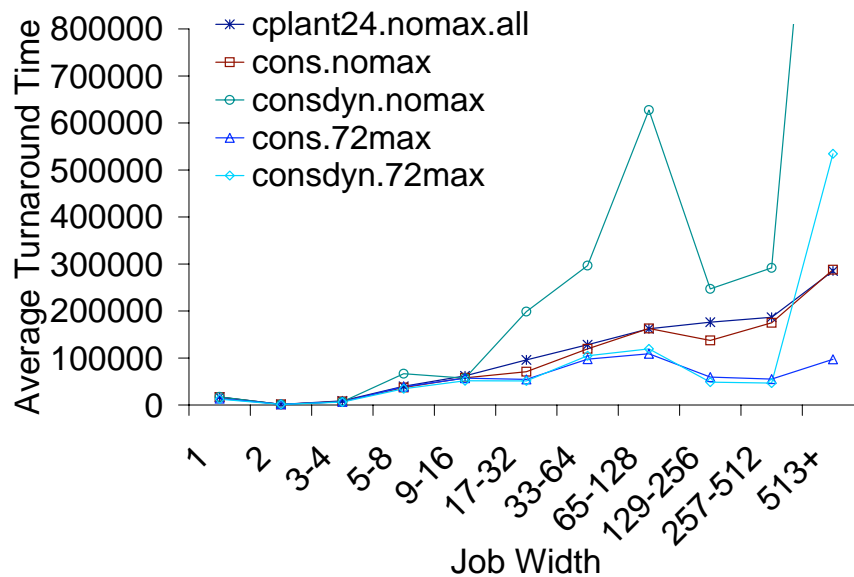


Figure 18: Average turnaround time for CPlant/Ross Simulations with conservative back-filling categorized by width

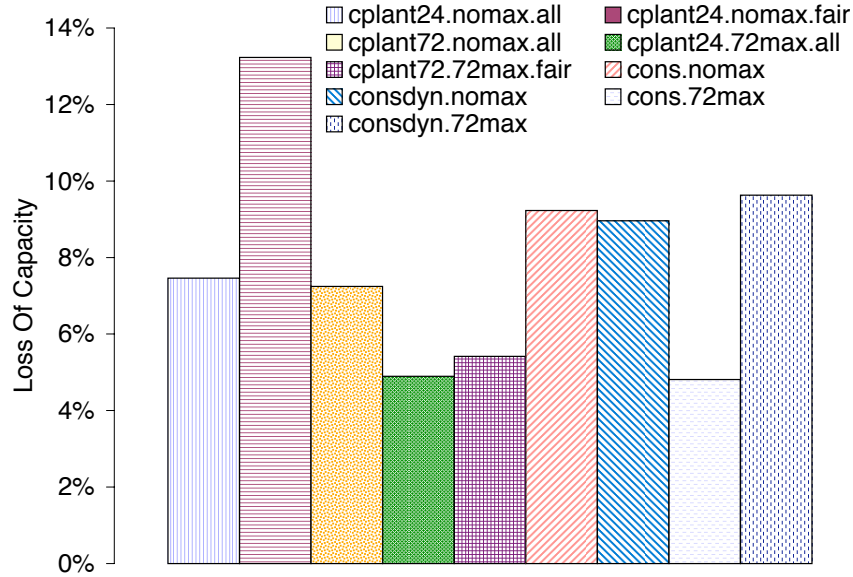


Figure 19: Loss of capacity for all CPlant/Ross simulations.

## 7 Conclusions

A CPlant workload trace was analyzed and presented. This trace was used to evaluate the fairness of the CPlant scheduler. Past fairness work was modified to accommodate a scheduling order considered “fair” in the Sandia environment. Scheduling modifications were introduced to improve fairness, average turnaround time, and loss of capacity.

A hybrid fairness metric is used to measure the fairness of the scheduling policies. The fairness metric is modified to utilize the “fairshare” queuing priority as the basis for social justice based fairness, as opposed to FCFS. The hybrid metric reduces the impact of the performance (as seen when using the CONS\_P metric) and the dependence on the current schedule (as seen when using a previous FST based metric). The fairness metric can be modified in a similar way to measure fairness via other alternative fairness priorities. This metrics allows for the analysis of unfairness by measuring the percentage of jobs that are treated unfairly and the average time that each submitted job misses the fair start time.

Several modifications to the CPlant scheduler were considered. Using a conservative backfilling schedule can help improve the fairness of wide jobs, which is important to super computing centers. Introducing 72 hour runtime limitations has the largest effect on fairness, loss of capacity and, average turnaround time.

## References

- [1] CPlant. <http://www.cs.sandia.gov/cplant/>. Computational Plant.

- [2] LSF. <http://www.platform.com/products/LSF/>. Platform Computing.
- [3] OpenPBS. <http://openpbs.org>.
- [4] SLURM. <http://www.llnl.gov/linux/slurm/>. A Highly Scalable Resource Manager.
- [5] Optimizing resource allocation. In *R & D Magazine*, page 49, September 2006.
- [6] Michael A. Bender, David P. Bunde, Erik D. Demaine, Sandor P. Fekete, Vitus J. Leung, Henk Meijer, and Cynthia A. Phillips. Communication-aware processor allocation on supercomputers. In *Springer Verlag, Lecture Notes in Computer Science, Vol. 3608*, 2005.
- [7] David P. Bunde, Vitus J. Leung, and Jens Mache. Communication patterns and allocation strategies. In *Proceedings of PMEO-PDS*, 2004.
- [8] S. H. Chiang and M. K. Vernon. Production job scheduling for parallel shared memory systems. In *Proceedings of International Parallel and Distributed Processing Symposium*, 2002.
- [9] D. G. Feitelson. Logs of real parallel workloads from production systems. <http://www.cs.huji.ac.il/labs/parallel/workload/>.
- [10] Dror Feitelson. Workshops on job scheduling strategies for parallel processing. [www.cs.huji.ac.il/~feit/parsched/](http://www.cs.huji.ac.il/~feit/parsched/).
- [11] Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong. Theory and practice in parallel job scheduling. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–34. Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
- [12] Mor Harchol-Balter, Karl Sigman, and Adam Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. In *IFIP WG 7.3 International Symposium on Computer Modeling, Measurement and Evaluation*, 2002.
- [13] Steven Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1162, pages 27–40. Springer-Verlag, Lect. Notes Comput. Sci., 1996.
- [14] David Jackson, Quinn Snell, and Mark Clement. Core algorithms of the Maui scheduler. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 87–102. Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.

- [15] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Technical Report EC-TR-301, Digital Equipment Corporation, September 1984.
- [16] J.P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In *5th Workshop on Job Scheduling Strategies for Parallel Processing*, 1999.
- [17] R. Kettimuthu, V. Subramani, S. Srinivasan, T. B. Gopalsamy, D K Panda, and P. Sadayappan. Selective preemption strategies for parallel job scheduling. In *Proc.of Intl. Conf. on Parallel Processing*, 2002.
- [18] Susan D. Klaviva. Department of energy does not effectively manage its supercomputers. Technical Report GAO/RCED-98-208, United States General Accounting Office, 1998.
- [19] Richard C. Larson. Perspectives on queues: Social justice and the psychology of queueing. *Operations Research*, 35(6):895–905, November 1987.
- [20] Vitus J. Leung, Esther M. Arkin, Michael A. Bender, David Bunde, Jeanette Johnston, Alok Lal, Joseph S. B. Mitchell, Cynthia A. Phillips, and Steven S. Seiden. Processor allocation on CPlant: archieving general processor locality using one-dimensional allocation strategies. In *Proceedings of Cluster*, pages 296–304, 2002.
- [21] Vitus J. Leung, Cynthia A. Phillips, Michael A. Bender, and David P. Bunde. Algorithmic support for commodity-based parallel computing systems. Technical Report SAND2003-3702, Sandia National Laboratories, October 2003.
- [22] David Lifka. The ANL/IBM SP scheduling system. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.
- [23] A. W. Mu’alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. In *IEEE Transactions on Parallel and Distributed Systems*, volume 12, pages 529–543, 2001.
- [24] D. Raz, H. Levy, and B. Avi-Itzhak. A resource-allocation queueing fairness measure. In *Proceedings of Sigmetrics 2004/Performance 2004 Joint Conference on Measurement and Modeling of Computer Systems*, pages 130–141, New York, NY, June 2004. Also appears as *Performance Evaluation Review Special Issue* 32(1):130-141.
- [25] Gerald Sabin, Garima Kochhar, and P. Sadayappan. Job fairness in non-preemptive job scheduling. In *International Conference on Parallel Processesing*, 2004.
- [26] Gerald Sabin and P. Sadayappan. Analysis of unfairness metrics for space sharing parallel job schedulers. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 2005.

- [27] Gerald Sabin, Vishvesh Sahasrabudhe, and P. Sadayappan. On fairness in distributed job scheduling across multiple sites. In *Proceedings of Cluster*, 2004.
- [28] Joseph Skovira, Waiman Chan, Honbo Zhou, and David Lifka. The EASY - LoadLeveler API project. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 41–47. Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.
- [29] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Selective reservation strategies for backfill job scheduling. In *8th Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002.
- [30] Sangsuree Vasupongayya and Su hui Chiang. On job fairness in non-preemptive parallel job scheduling. In *Parallel and Distributed Computing and Systems (PDCS)*, number 17. IASTED, November 2005.

# UNLIMITED RELEASE

## INITIAL DISTRIBUTION:

- 1 State University of New York  
Dept. of Computer Science  
Attn: M. A. Bender  
Stony Brook, NY 11794-4400
- 1 Knox College  
Computer Science  
Attn: D. P. Bunde  
Galesburg, IL 61401
- 20 The Ohio State University  
Dept. of Computer Science and Engineering  
Attn: G. Sabin (10)  
P. Sadayappan (10)  
Columbus, OH 43210-1277
- 1 MS 0321 J. L. Mitchiner, 1430
- 1 0321 J. S. Peery, 1400
- 1 0370 J. H. Strickland, 1433
- 1 0376 S. J. Owen, 1421
- 1 0378 R. M. Summers, 1431
- 1 1138 J. R. Johnston, 6325
- 1 1316 M. D. Rintoul, 1412
- 1 1318 K. F. Alvin, 1318
- 10 1318 V. J. Leung, 1415
- 1 1318 C. A. Phillips, 1412
- 1 1318 S. L. K. Rountree, 1415
- 1 1318 J. R. Stewart, 1411
- 1 1318 D. E. Womble, 1410
- 1 1319 J. A. Ang, 1422
- 1 1319 N. Pundit, 1423
- 1 1319 J. R. Stearley, 1422
- 1 1320 S. S. Collis, 1416
- 1 1322 J. B. Aidun, 1435
- 1 1322 S. S. Dosanjh, 1420
- 1 1323 D. H. Rogers, 1424
- 1 0899 Technical Library, 9536 (electronic copy only)

Intentionally Left Blank