Fair Share on High Performance Computing Systems: What Does Fair Really Mean?

Stephen D. Kleban
Sandia National Laboratories
Albuquerque, NM 87545
sdkleba@sandia.gov

Scott H. Clearwater
P.O. Box 620513
Woodside, CA 94062
clearway@ix.netcom.com

Abstract

We report on a performance evaluation of a Fair Share system at the ASCI Blue Mountain supercomputer cluster. We study the impacts of share allocation under Fair Share on wait times and expansion factor. We also measure the Service Ratio, a typical figure of merit for Fair Share systems, with respect to a number of job parameters. We conclude that Fair Share does little to alter important performance metrics such as expansion factor. This leads to the question of what Fair Share means on cluster machines. The essential difference between Fair Share on a uni-processor and a cluster is that the workload on a cluster is not fungible in space or time. We find that cluster machines must be highly utilized and support checkpointing in order for Fair Share to function more closely to the spirit in which it was originally developed.

1. Introduction

The Advanced Simulation and Computing Initiative (ASCI) supercomputers were developed for running high-fidelity, full physics, predictive codes for structural, electrical, and thermal simulations. In order to satisfy the needs of this program it is necessary for the jobs running on these machines to be processed in a timely manner in order to meet project milestones. The task of the ASCI facility manager is made all the more difficult by the fact that the distribution of usage is highly skewed to a few users running a number of very large jobs[1].

Fair Share is a widely used queueing algorithm for prioritorizing jobs on the basis of a "share" of the machine and past and current usage. Using log data we performed a number of analyses involving correlations among jobs sizes, expansion factors, service ratios, waiting times, and number of CPUs.

The key contribution of this paper is that we show that Fair Share is not playing a large role in prioritorizing jobs. In particular, in terms of the quality of service metric expansion factor, we found that the expansion factor is not strongly influenced by the administrative value of shares in the sense that over-subscribers can have systematically low expansion factors over an extended period of time. This lack of a systematic effect of Fair Share on this system leads us to question what fair really means. We find that because jobs on a cluster are not flexible in terms of space (number of processors) and time (not checkpointable), Fair Share is not able to achieve real-time fairness as it can on a uni-processor. Rather, uni-processor-like fairness on a cluster is achievable only with long time horizons.

2. Environment

In this section we describe the particular environment we used in this study, including the machine, the queuing algorithm, the data, and simulator.

2.1 Supercomputer

The machine used in this study is Blue Mountain located at Los Alamos National Laboratory (LANL). Blue Mountain has 6144 250-MHz CPUs in 48 shared memory multi-processor "boxes" of 128 CPUs (126 usable by jobs) each and covering 10,000 square feet of floor space. The machine was commissioned in 1998 and has a peak capacity of Rpeak = 3TF/s and according to www.top500.org ranks 15th in the world as of June 2002 in terms of Rmax.

2.2 Queue Algorithm

Blue Mountain relies on the Load Sharing Facility (LSF) for queue management[2]. Fair Share has also been applied to massively parallel supercomputers, specifically at LANL running LSF and at Lawrence Livermore National Laboratory (LLNL) running Distributed Production Control System (DPCS)[3].

Fair Share for job scheduling was originally designed for managed resource allocation of processes on a time-sharing uni-processor system[4]. One problem these priority-based schemes have is that their parameters are determined ad hoc. For example, the employment of time-decayed penalties in adjusting the priority involves the assumption of the time scale over which "fairness" is to be obtained. Additionally, Fair Share "assumes a fixed workload consisting of long-running compute-bound processes to ensure steady-state fairness"[5]—a situation rarely encountered with ASCI supercomputer clusters.

Fair Share, as implemented on Blue Mountain, assigns each job a priority based on the submitter's group priority as well as the current and recent past usage by that group. Priorities are periodically updated and the queue resorted based on the latest usage. Fair Share also utilizes a backfill mechanism whereby jobs that are not the highest priority can run if there are sufficient processors and their running time (based on the user's estimate) will not delay the running the current highest priority job.

There are significant differences between processes on a uni-processor time-sharing system and jobs on a supercomputer as shown in Table 1, unlike the claims made in Kay and Lauder[4]. For example, *processes* can be freely given more or less of the processor, but a *job* on a supercomputer is not fungible into an arbitrary number of processors—it must run a specific number of processors. Also, jobs on ASCI machines are not system checkpointable and so cannot be arbitrarily swapped in and out to satisfy Fair Share allocations. The key difference of how Fair Share runs on ASCI machines is that it does not cap usage as is done on a single processor with processes. In other words, on the processor, or time-sharing version of Fair Share there is a hard cap on what fraction of the machine a process can have. On Fair Share for clusters such as ASCI, no such hard cap exists. As we will show, there is only a degradation of subsequent job priority that does not in practice significantly impact individual or overall group level performance over time.

Table 1. Fair Share on Uni-processors and Supercomputers

	Uni-processor	ASCI machines
Unit of resource being shared	process	job
Prioritized once running?	yes	no
Fungible amount of processors?	yes	no
System Checkpointable?	yes	no
Can get more of the machine than your share when machine is highly loaded?	no	yes

Further, the analysis in Kay and Lauder says "a non-Poisson distribution probably indicates problems, such as a class of users (not necessarily in the same group) that are consuming a disproportionately large amount of the resources."[4] As we have previously reported, ASCI machine usage is dominated by the "problem" of a few large users and the submittal times and job sizes are fat-tailed (at the high end) as well[1,6].

2.3 Data

We have the choice of using log data or generating synthetic data for our analysis. Because Blue Mountain relies on a hierarchy of users and the fact that a number of jobs are dependent on one another, it is important to retain these correlations. Consequently we decided to use log data. Also, Blue Mountain is divided into two partitions, one for smaller jobs and one for larger jobs. We focused our analysis on the large partition which has 4662 CPUs divided into 37 boxes with 126-CPUs each. Nearly all jobs are a multiple of 126, the exception being debug jobs. The logged data were taken over a period of 83 days and contained 8,171 jobs. Within the large partition there are several queues. We focused on the "large queue" (largeq) which is Fair Shared and had 7011 jobs. The maximum run time on largeq is 12 hours. Errors in the log due to missing or acausal (e.g., the dispatch occurring before the submission) were not used and occurred at a level of a few percent.

2.4 Simulation

In this analysis we use the Big Iron Resource Management simulator (BIRMinator)[7]. Briefly, BIRMinator uses synthetic or log data in conjunction with a queue description, user definitions, and machine definitions to simulate job submission, dispatch and completion. The simulation takes into account the queue parameters, job dependencies, and group share hierarchies. We created a separate set of job data for each study we did corresponding to a particular job parameter being scaled.

3. Analysis

In this section we analyze the log data with respect to shares, usage, service ratio (actual usage/shares), and expansion factor (1+wait/run).

3.1 Group Shares

The purpose of group share analysis is to see how, or if, group shares influence the expansion factor (a "quality of service" measure) for jobs of the ASCI supercomputers. Group shares are the scaling factor for job priorities on the Fair Share system. The shares are determined by administrators and user group representatives and may change over time. Essentially, the shares provide a bias in the prioritization of jobs, i.e., groups with more shares have a higher priority, all other factors being equal.

LSF at Blue Mountain uses group level Fair Share. Using user/group information we were able to identify which group(s) a user belonged to in the large queue group hierarchy. It is important to note that the 18 largest users out of hundreds, accounted for 93% of the "largeq" usage. The shares used by these users accounted for 54% of the shares. The 93% usage by only half the share groups is a telling indicator of how poorly the shares were aligned with the actual usage during this period. Often the figure of merit of system administrators is to equate shares with usage. However, the mismatch we see here is not surprising given that "proper" share allocation on a less than fully utilized machine requires omniscience of future usage which is difficult due to the natural fluctuations in activity that occur in projects. This makes the task of the system administrator all the more difficult because their figure of merit is the "service ratio" which is the actual usage normalized by the share allocation.

Table 2 shows the relationship between shares and usage. Note that because of double counting from multiple group membership this column will add up to be slightly over 100%. Note that the high share groups do not correlate well with the high usage groups. The "top-level parent groups" are the highest level group within a Fair Share hierarchical "bank" of users.

Table 2. Group Share and Usage

Group	Share	Usage
g2	.050	.056
g4	.050	.158
g8	.050	.210
g15	.050	.010
g20	.060	.028
g21	.120	.067
g22	.060	.213
g23	.060	.416
g29	.360	.023
g30	.0225	.000
g31	.0675	.010
g39	.050	.018
g68†	.250	.723
g69†	.300	.477
g70†	.450	.033

†top-level parent group

This data, graphically shown in Fig. 1, shows the relationship between shares and actual usage for the non-parent groups. If shares were truly aligned with usage then we would expect the points to lie along a line with positive slope. However, as we see for the low share groups their usage is quite variable and there is no correlation.

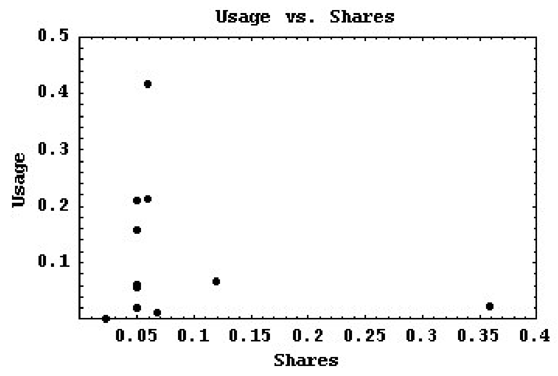

Fig. 1 Usage versus Shares. (Top-level parents not included.)

3.2 Wait Times

The wait time is probably the most important performance characteristic from the user's perspective. The wait time is defined to be the dispatch time minus the submit time for jobs that do not depend on another job. In cases where jobs are dependent on another job before they can run, the wait time is defined as the dispatch time minus the time when the independent job (meaning that it's dispatch to the machine is not dependent on another job finishing as specified by the user) finished. In Fig. 2 we show the relationship between the wait time in seconds and the job size measured in CPU-sec. Wait times of zero were set to 1 so the log calculation would give a finite result. Most of the values of wait are close to zero, with over half the jobs running within 10 seconds of when they were submitted. Thus for most of the jobs that run, the queuing system is not playing much of a role.

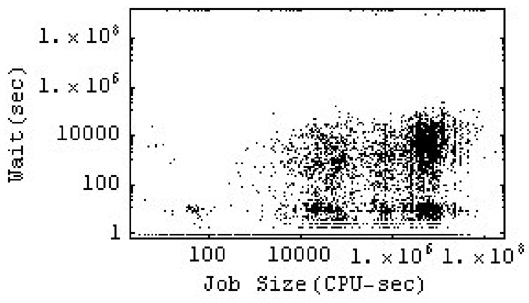

Fig. 2. Log$_{10}$(Wait time(sec)) versus Log$_{10}$(job size (CPU-sec)) for all the jobs in the large partition.

Nevertheless, a substantial number of jobs wait a very long time. For jobs about 100 CPU-sec long, the wait time was systematically lower than for larger jobs. For jobs larger than 1000 CPU-sec, the tail of the wait time distribution was nearly the same for all job sizes up to nearly 10^8 CPU-sec. Thus, the queuing system is not making a distinction between large and small jobs. This occurs in spite of the fact that job size is an integral part of the Fair Share calculation as it enters directly into the past and current usage and thence into priority.

However, the situation is quite different if we look at wait time versus the number of processors as seen in Fig. 3. Here we see a definite relationship between the wait time and number of processors with larger jobs waiting longer. This relationship makes intuitive sense because we would expect smaller jobs to be able to fit more often in backfill scenarios. Although there is considerable spread among the points, a best fit line gives a good visual fit to the relationship:

$$wait = 155 \times CPU^{1.44} = 3 \times 10^{7} f^{1.44}$$

where *wait* is in seconds and *CPU* is the number of processors used by the job and *f* is the fraction of processors used by the job. Thus, doubling the size of the job increases the wait time by an average of $2^{1.44} = 2.7$.

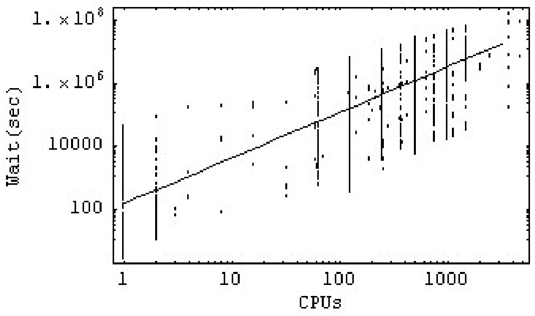

Fig. 3. Wait versus CPU with best fit line.

The power law relationship found above illustrates an important issue in large scale computing. Namely, these machines are designed to run enormous jobs, but still there is a diminishing of returns for larger jobs because the wait time grows faster than the number of CPUs.

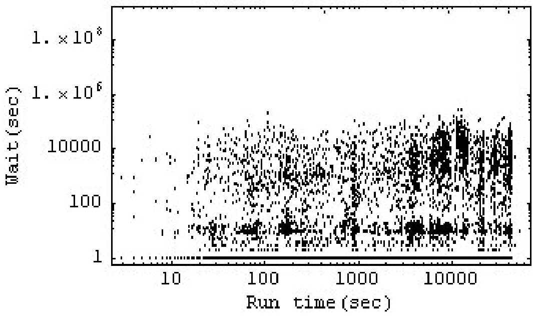

Fig. 4. Wait versus Runtime.

Fig. 4 shows the relationship between wait time and run time and no correlation is seen. Thus, the relationship between wait time and CPUs indicates that the finite number of processing resources is the critical element in how long a job waits and depends less on how long it will reside in the machine.

3.3 Expansion Factor

Expansion factor is a measure of quality of service typically calculated as (*runtime* + *waittime*)/*runtime*, but for this analysis the expansion factor was calculated for each group according to the formula:

$$EF_{group} = \frac{\sum \left(\frac{run + wait}{run} \right) weight}{\sum weight} = \frac{\sum \left(1 + \frac{wait}{run} \right) weight}{\sum weight}$$

where *weight* is either *jobsize = CPUs*× *run* or number of *CPU*s, and *wait=dispatch – submit*. The weighting factor of jobsize was used to reduce the effect of small jobs that waited a long time from overly biasing the results.

Also, note that there are cases where certain jobs are dependent upon other jobs. This means even if such a dependent job were submitted at the same time as the job it depends on, the dependent job will have an anomalously large wait time. This will be true even if the dependent job runs immediately after the job it depends on finishes. Thus we modify the *EF* for dependent jobs to be:

$$EF_{group,dep} = \frac{\sum \left(1 + \frac{dispatch - finish_{independent}}{run} \right) weight}{\sum weight}$$

where $finish_{independent}$ is the finish time of the independent job.

The data in Table 3 represents the share groups and the top-level parent groups along with their EFs. In cases where users did belong to more than one group, the contribution to the expansion factor calculation were made equally to each of the groups and is thus a source of error because we cannot uniquely invert the log to determine from which group a job was submitted. Unless otherwise noted, we use the job size weighted expansion factor in the analyses that follow rather than the CPU weighted expansion factor.

As can be seen clearly from Table 3, the expansion factor does not correlate well with the shares. It may seem that groups with lower shares would have a higher expansion factor because they should wait longer, but there appears to be no strong correlation between shares (Fig. 5). For actual usage (Fig. 6) there does seem to be a correlation with the exception of two groups that tended to submit large jobs in bunches.

Table 3. Shares and Expansion Factors

Group	Share	expansion factor (CPU weighted, dependency corrected)	Avg. Utilization at Submit time
g2	.050	1.35	.87
g4	.050	13.53	.88
g8	.050	9.57	.88
g15	.050	2.61	.75
g20	.060	21.6	.90
g21	.120	2.45	.89
g22	.060	14.93	.87
g23	.060	10.68	.87
g29	.360	94.90	.77
g30	.0225	1.35	.94
g31	.0675	2.23	.93
g39	.050	4.00	.89
g68†	.250	12.03	.87
g69†	.300	9.58	.88
g70†	.450	70.9	.85

†top-level parent group

We should also point out that an examination of the utilizations at submit time were nearly identical for the groups indicating that there was not a systematic effect influencing the results. For example, a group was not waiting for the machine to be relatively free before submitting thereby obviating the effect of Fair Share.

The relationship between the expansion factor, CPUs, and run time on a job-by-job basis is shown in Fig. 7. The discreteness in the CPU axis is due to the fact that users using the largeq (most of the jobs in the large partition are from the largeq) must request processors in multiples of 126 CPUs. Intuitively, we might expect a nice surface rising from the smaller jobs (near the origin) to the larger jobs (right rear upper corner), but that does not appear to be the case.

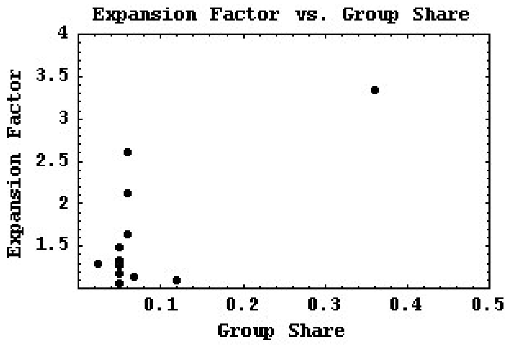

Fig. 5. expansion factor(weighted by job size) corrected for dependent jobs versus Group Share.

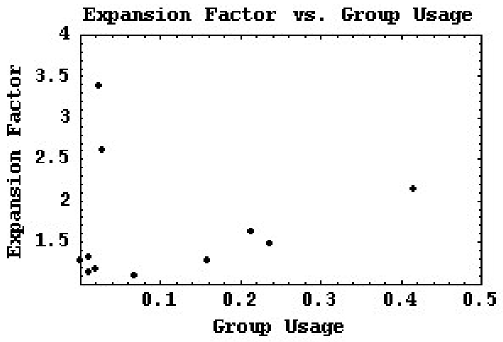

Fig. 6. expansion factor(weighted by job size) versus Group Usage.

Looking at just the relationship between expansion factor and CPU, we see from Fig. 8 that the relationship is not as clear as it is for wait time versus CPU as in Fig. 3. Recall that expansion factor is 1+*wait/run* so that the runtime must be washing out the effect we found in Fig. 3. Thus large jobs, in terms of CPUs, are neither favored nor discriminated against in terms of expansion factor.

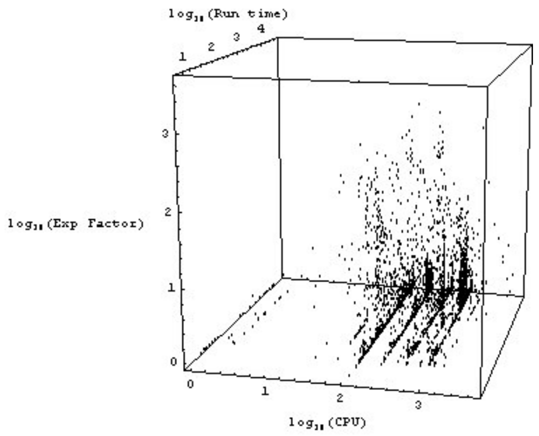

Fig. 7. Expansion Factor versus CPU versus Run time.

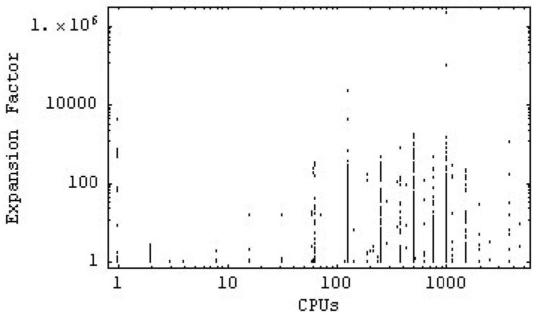

Fig. 8. Expansion Factor versus CPU.

3.4 Service Ratio

Service Ratio is considered by facility managers at Blue Mountain as an important figure of merit. Basically, Service Ratio, SR, is the ratio of cycles used by a share group to the shares actually allocated. The ideal value for the Service Ratio is 1 which means that each share group is using exactly the fraction of cycles represented by their relative shares. The problems associated with such a figure of merit is the omniscience required. The unavoidable fluctuations in projects over time make it next to impossible to achieve an SR=1. The Service Ratio is the cumulative usage by a group divided by its assigned share of the machine over the entire time period.

For completeness, Table 4 shows the relationship between shares and Service Ratios. As expected there is very little correlation between the two.

Table 4. Shares and Service Ratios

Group	Share	Service Ratio
g2	.050	1.12
g4	.050	3.16
g8	.050	4.76
g15	.050	0.19
g20	.060	0.47
g21	.120	0.55
g22	.060	3.55
g23	.060	6.93
g29	.360	0.06
g30	.0225	0.00
g31	.0675	0.15
g39	.050	0.36
g68†	.250	2.89
g69†	.300	1.59
g70†	.450	.073

†top-level parent group

Fig. 9 shows the expansion factor versus the Service Ratio for the lowest level groups (leaf nodes in the group hierarchy). Again there is no definitive correlation between expansion factor and Service Ratio. However, there does seem to be some clustering around small Service Ratios and small expansion factors, which is precisely the intention of the Fair Share algorithm—meaning that if you are below your allocation you should wait less.

Yet, there is by no means a definitive relationship, as some low Service Ratio groups have a high expansion factor and some high Service Ratio groups have a low expansion factor. In cases where one user dominates the usage in a share group, all the other users in that group feel the effect through lower priority for their jobs, although this is not a big effect in practice. This is a consequence of using Fair Share at the group level rather than at the user level.

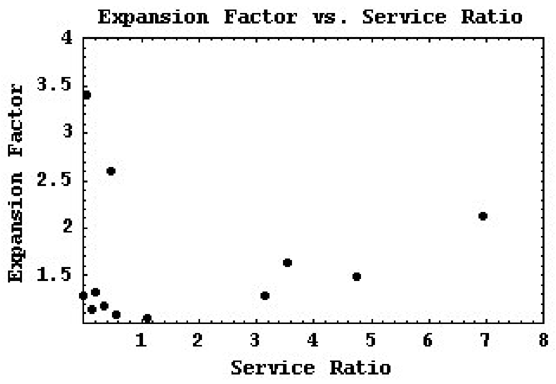

Fig. 9. Expansion Factor(weighted by job size) versus Service Ratio.

Fig. 10 shows the relationship between Service Ratio and shares and usage. There is a tendency for larger Service Ratios to be associated with larger usage, but any trend is less obvious for shares—the more important administratively defined parameter. Note that the Service Ratios are computed at the end of the 84-day run as are the usages.

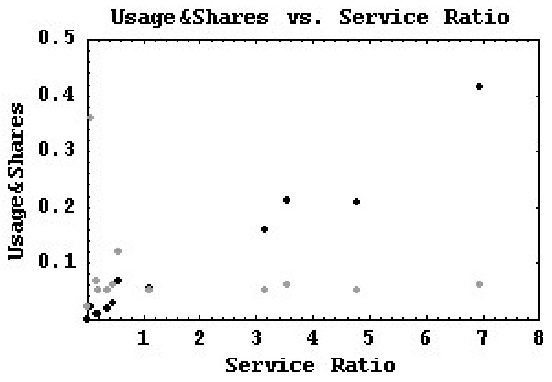

Fig. 10. Shares and Usage versus Service Ratio.

Fig. 11 shows a plot of jobsizes versus Service Ratios versus expansion factors. The plot has three main features:

1) Main body: {10^4<JS<10^7, SR<2, 1<EF<4} Most of the jobs fall into this category. The jobs in this region show little correlation between JS, SR, or EF.

2) Large SR appendage: {10^6<JS<10^7, SR>2, 1<EF<4}. These jobs are from heavy users who are far beyond their Service Ratio target of 1 and yet have lower than average EFs. These may represent opportunistic users who take advantage of lower utilization periods such as evenings and weekends to get their large job work done. Indeed, an inspection of the log data shows that higher SR jobs tend to be after normal work hours and during the weekend.

3) Large EF appendage: {10^4<JS<10^5, SR<2, 10<EF<1000} These are small to moderate jobs with

typical SR values and yet have very long expansion factors. Examination of the log reveals that these jobs tend to be week day daytime users who are competing for resources during high utilization periods.

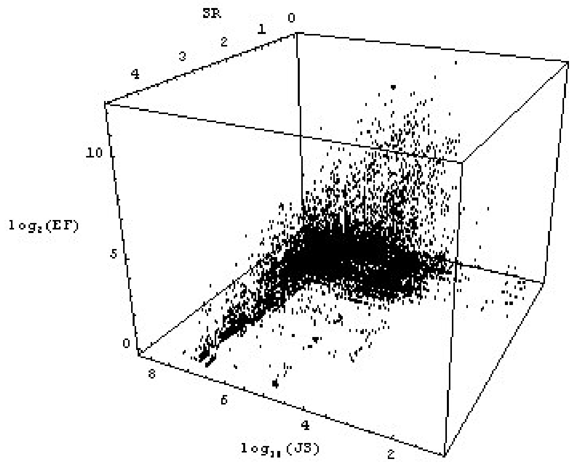

Fig. 11. Job size(JS) versus Service Ratio (SR) versus expansion factor (EF).

3.5 Spearman Rank-Order Analysis

In order to see if there is a correlation between the various job and performance parameters, we performed a Spearman rank-order correlation test[8]. The advantage of the Spearman test is that it is non-parametric and depends only on the relative ordering of two lists. Basically, the Spearman test compares the relative ordering of two lists to see how well they line up for the purpose of determining their correlation. It also turns out that there is a relationship between the Spearman rank-order statistic and a statistical confidence level.

Specifically, we tested to see how well the ranks of the various columns in Tables 2-4 are correlated for the non-parent groups. Comparing the ranks of Share with Usage (Table 2), Share with expansion factor (Table 3), and Share with Service Ratio (Table 4) showed no correlation to a high level of statistical confidence. Ranks of expansion factor with Usage and Service Ratio also showed no correlation. The only significant correlation was between Usage and Service Ratio. These results are shown in Table 5 which gives the two ranks and the significance of a non-zero rank. A high value (>>0 and ~1) of significance means there is not a significant correlation and a low value (~0) means there is a significant correlation.

The conclusion we draw from this rank analysis is that relative ranks are not preserved across various performance metrics with respect to shares. This does not help the case that shares, usage, and service ratio are importantly correlated by way of the queuing system as regards the performance of the machine.

Table 5. Rank Analysis

Rank List 1	Rank List 2	Significance
Shares	Usage	0.508
Shares	EF	0.422
Shares	Service Ratio	0.946
Usage	Service Ratio	0.000

4. Discussion

We have analyzed an instance of a Fair Share queue algorithm on a large cluster. In general we find that Fair Share is not playing a large role in prioritizing jobs. We base this on a number of findings that are discussed below.

In terms of the quality of service metric expansion factor, what we conclude from the analysis of the log data is that the expansion factor is not strongly influenced by the administrative value of shares in the sense that over-subscribers can still have systematically low expansion factors.

We did find a superlinear relationship between the wait time and the number of CPUs. This effect did not carry over to the relationship between expansion factor and number of CPUs.

In terms of the relationship between jobsize, Service Ratio, and expansion factor (Fig. 11), the largest jobs were spread out over all Service Ratios and tended to have a lower expansion factor with higher Service Ratios(e.g., large evening and weekend jobs). Smaller jobs tended to have larger expansion factors(e.g., weekday day jobs).

We also performed a series of Spearman rank order tests on the job parameters. Comparing the ranks of Share with Usage, Share with expansion factor, and Share with Service Ratio showed no correlation to a high level of statistical confidence. Ranks of expansion factor with Usage and Service Ratio also showed no significant correlation. The only significant correlation was between Usage and Service Ratio. The conclusion we draw from this rank analysis is that relative ranks are not preserved across various performance metrics with respect to shares.

We also found that the difference in the job run order, as measured by the order that the job was submitted compared to the order that it was dispatched to the machine for running, was generally small indicating that most jobs ran close to the order they were submitted.

In summary, we are now in a position to answer the question “what does fair really mean” for cluster computing. The answer to this question is complicated because fairness seems to be in the mind of the beholder.

For example, we should note that Fair Share on this cluster is not being used in the same way as it was originally intended for single CPU systems with "cost-free" process migration. Consequently, "fair" on a uni-processor necessarily has a different meaning than on a supercomputer without checkpointing.

Fairness on a uni-processor is achievable in principle over all time horizons because the processes can be swapped in and out freely according to the frequency that the Fair Share algorithm updates. Such is not the case on clusters because cluster jobs must run on a specific number of processors and run to completion on those processors once started. Thus, the implications of this crucial difference is that the time horizon over which the Service Ratio targets are to be met on the cluster is much longer. In principle, uni-processor-like Fair Share-ness is asymptotically achievable on clusters over an infinite time horizon.

Theoretically speaking, as long as all the cluster users have jobs in the queue, then over a sufficiently long time horizon they will all reach a Service Ratio of 1.

On a supercomputer such as Blue Mountain, Fair Share is not playing a big role because the machine is run at a sufficiently low usage, insuring timely and effective turnaround. Fair Share's main role on Blue Mountain is essentially as a moderator for cases of queue flooding.

Our results are not meant to be an indictment of Fair Share systems, but rather an indication that its role in clusters without checkpointing is quite different from that found on uni-processors. Our conclusions should apply to other large grids that use Fair Share with groups and run at utilizations of about 80% or less on average, as is the case with Blue Mountain. For higher utilizations, Fair Share behavior on a cluster approaches that of a uni-processor., albeit with a much time horizon.

Grid operators need to distinguish between the complementary goals of rapid turnaround and running the machine at maximum utilization. To achieve the first requires a "spare capacity" or "over design" or "under subscription." While seemingly wasteful of resources, this strategy of spare capacity is inseparable from the need for accommodating the very large jobs that are the raison d'être for grids such as the ASCI machines. In other words, the worth of the grid is not necessarily in being able to simply supply a large compute resource over time, but to have that supply readily available on short notice.

From a more analytic perspective, spare or reservoir capacity can be shown to be to vital to handling clustering of jobs that occur over time leading to queue floods and droughts[9].

If the goal is to make Fair Share on grids perform as on a uni-processor, then gang-scheduling could be used[10]. However, has its own set of issues involving what in many cases is a significant cost of migration, unlike that of processes on a uni-processor.

Another possible alternative to fairly sharing supercomputer resources is to employ a market-based approach in which the users themselves determine the priority of their jobs (within some limits)[11]. This approach empowers users to determine the relative priority of their own jobs rather than having it decided by an algorithm.

Acknowledgements

This paper would not have been possible without the expertise of many people who provided us with detailed knowledge of algorithms, data formats, as well as providing the job log data. The authors gratefully acknowledge the assistance of Stephany Boucher, Charles Hales, Tom Klingner, Jerry Melendez, and Randall Rheinheimer. We also wish to thank the anonymous reviewers for their helpful comments. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

References

[1] S. H. Clearwater and S. D. Kleban, "Heavy-Tailed Distributions in Supercomputer Jobs", Sandia National Laboratories SAND2002-2378C (2002).
[2] LSF Batch Administrator's Guide, Sixth Edition, Platform Computing Corporation (1998).
[3] DPCS Reference Manual (unpublished).
[4] J. Kay and P. Lauder, "A Fair Share Scheduler" *Communications of the ACM*, vol. 31 pp.44-55 (1988).
[5] C. A. Waldspurger and W. E. Weihl, "Lottery Scheduling: Flexible Proportional-Share Resource Management" in *Proceedings of the First Symposium on Operating Systems Design and Implementation*, November, (1994), pp.1-11.
[6] S. H. Clearwater and S. D. Kleban "Relaxation Phenomena in Supercomputer Jobs", http://www.arxiv.org/abs/cond-mat/0208531 (2002).
[7] S. D. Kleban and S. H. Clearwater, "A Big Iron Resource Management Simulation System" in *Proceedings of the Advanced Simulation Technologies Conference 2002*, San Diego, CA 14-18 April (2002).
[8] Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., *Numerical Recipes in C* Cambridge, New York (1988).
[9] S. D. Kleban andS. H. Clearwater, in preparation.
[10] D. G. Feitelson and M. A. Jette, "Improved Utilization and Responsiveness with Gang Scheduling" In *Job Scheduling Strategies for Parallel Processing* D. G. Feitelson and L. Rudolph (eds.) *Lecture Notes Computer Science* Vol. 1291 (1997), pp. 238-261.
[11] S. Kleban and S. Clearwater, "A Market-Based Architecture for Supercomputer Resource Management" in *Proceedings of the ACM/IEEE Super Computing 2001*, Denver, CO, Nov 10-16 (2001).