HAL
open science

Analysis of the Jobs Resource Utilization on a Production System

Joseph Emeras, Cristian Ruiz, Jean-Marc Vincent, Olivier Richard

To cite this version:

Joseph Emeras, Cristian Ruiz, Jean-Marc Vincent, Olivier Richard. Analysis of the Jobs Resource Utilization on a Production System. Job Scheduling Strategies for Parallel Processing, 2013, Boston, United States. hal-00918372

HAL Id: hal-00918372

https://inria.hal.science/hal-00918372v1

Submitted on 13 Dec 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Analysis of the Jobs Resource Utilization on a Production System

Joseph Emeras, Cristian Ruiz, Jean-Marc Vincent, Olivier Richard

LIG Laboratory, Grenoble, France
<Joseph.Emeras, Cristian.Ruiz, Jean-Marc.Vincent, Olivier.Richard>@imag.fr

Abstract. In *HPC* community the System Utilization metric enables to determine if the resources of the cluster are efficiently used by the batch scheduler. This metric considers that all the allocated resources (memory, disk, processors, etc) are full-time utilized. To optimize the system performance, we have to consider the effective physical consumption by jobs regarding the resource allocations. This information gives an insight into whether the cluster resources are efficiently used by the jobs. In this work we propose an analysis of production clusters based on the jobs resource utilization. The principle is to collect simultaneously traces from the job scheduler (provided by logs) and jobs resource consumptions. The latter has been realized by developing a job monitoring tool, whose impact on the system has been measured as lightweight (0.35% speed-down). The key point is to statistically analyze both traces to detect and explain underutilization of the resources. This could enable to detect abnormal behavior, bottlenecks in the cluster leading to a poor scalability, and justifying optimizations such as gang scheduling or besteffort scheduling. This method has been applied to two medium sized production clusters on a period of eight months.

Keywords: Workload Traces; Monitoring; Performance Evaluation; Optimization; High Performance Computing

1 General Context

High Performance Computing (HPC) platforms have appeared as a solution for solving advanced computation problems. Nowadays these systems have evolved passing from the shared memory multiprocessors to clusters which can have hundreds of thousands of processors. In these kind of systems a central component called the Resource and Job Management System (RJMS) is in charge of managing the users' tasks (jobs) on the system's computing resources.

Studying the RJMS workload has become a widely used method for HPC systems valuation. The workload in the RJMS context can be defined as the set of all individual jobs that are processed by the system during a specific period of time. With workload traces, one can reconstruct the scheduling, determine the system's computing resources utilization, compare different systems and their

workloads according to several metrics like *Average Weighted Wait Time* and *Average Weighted Response Time* as used in [1]. This study can lead to the construction of models as proposed in [2] and more generally described in [3]. However, looking only at the workload traces may not be sufficient. To get a better understanding of the use of such systems we need to look at both how the jobs interact with the RJMS but also how they consume the allocated resources. In other words, we need to look at both the workload and the jobs activity on the resources.

The idea of associating RJMS workload traces to the jobs resource consumptions has been mentioned in several works. [4] points out the fact that the System Utilization metric, commonly used in system evaluation misses information about the computer sub-systems (network, memory, processor) usage. According to [5] it is also necessary to take into account other characteristics such as I/O activity which have a big impact on the global performance of HPC systems. However, the aforementioned association has never been fully achieved.

The System Utilization metric is the ratio of the computing resources allocated to the jobs over the available resources in the system. In fact, this metric corresponds more to the *System Resource Allocation* as it does not give information about the physical utilization of the resources. In this paper we will focus on the Resource Utilization metric which reflects the ratio of the consumption of a resource by a job over the amount of resource allocated by the system to this job. For each resource type: core, memory, IO, we will look at their utilization by the jobs on the clusters.

The following Section presents several works related to Workload and Jobs tracing. The first part presents briefly D. Feitelson et al. works on workload traces then several methods for retrieving information about the jobs consumptions are presented. Then in Section 3 we present the solution chosen to collect jobs resource consumptions data. Section 4 describes the clusters on which data was collected and the trace characteristics. Section 5 presents the analysis of the results of the different resources consumption metrics. Finally, Section 6 discusses the results and gives some perspectives.

2 State of the Art

2.1 Workload Traces

Workload traces are provided by the Resource and Job Management System, they contain information about the jobs arrivals, their resources request and allocation, their characteristics like runtime, wait time or execution information. Two workload traces format exist and are commonly used:

a) Standard Workload Format (SWF)[1]. It is an initiative to make workloads data on parallel machines freely available and presented in a standard format[6]. This work is presented in the Parallel Workload Archive[2]. It provides

[1] SWF: http://www.cs.huji.ac.il/labs/parallel/workload/swf.html
[2] PWA: http://www.cs.huji.ac.il/labs/parallel/workload

several traces from real production systems and is a big step in the field of workload characterization. Many other works were based on this format, in particular for workload models generation. The SWF format gives information about the jobs requests, allocations, characteristics and consumptions. The jobs consumptions values provided by SWF are the average processor time and the average memory used by processor.

b) Grid Workload Archive (GWA)[3]. The Grid Workload Archive constitutes an effort to build a data repository of grid workload traces for the scientific community. This format is based on SWF, it adds some grid specific aspects like sites, virtual organizations, co-allocations, workflows. This information contributes to the improvement of the middleware in charge of the scheduling process.

2.2 Jobs Consumption Traces

Two options are possible to produce a trace of the jobs consumptions. First, giving for each job, for each type of resource (memory, processor, disk, network), a single representative value of the resource consumption. The question is thus *"How representative of the real consumption this value is?"*. Or, for each job, giving a trace of the resource consumptions over time. This can be viewed as a monitoring of the jobs consumptions.

In [7], R. Jain proposes one possible classification of the different monitoring techniques. In this classification, the monitoring process can be *event-driven* or by *sampling*. The event-driven method is very efficient and there is no monitoring overhead if the event is rare. The sampling method as for it, is well adapted for frequent events but a loss of data captured is inevitable and a frequency resolution has to be chosen. This section presents the state of the art of the existing monitoring systems that can be adapted to our context.

Monitoring provided by the existing batch schedulers. Several RJMS provide an embedded system for monitoring the jobs consumptions.

SLURM [8][4]: uses different mechanisms to know which processes are members of a SLURM job, then monitors their consumptions. The cgroup isolation mechanism requires a very recent kernel version, the other one is based on the Linux process table.

OAR [9][5]: provides a mechanism that monitors the jobs consumptions in terms of processor, memory and network. As for SLURM, it uses different mechanisms to know which processes are members of an OAR job, these are the cpuset or cgroup features of the kernel and require a recent kernel version. This data collection is not automatic and is triggered at user request.

[3] GWA: http://gwa.ewi.tudelft.nl/pmwiki/
[4] SLURM. https://computing.llnl.gov/linux/slurm/
[5] OAR: http://oar.imag.fr

LoadLeveler[6]**:** allows the user to gather information about resource consumption. It offers different ways to consult this information and create different class of reports. As for OAR, this mechanism is not automatic.

All these approaches are linked to a particular batch scheduler system. For all of them, the traces generated give for each job a single value (generally the mean) for each type of resource consumption. This is problematic for several reasons. First for the memory consumption, the mean does not give valuable information. At least the maximum value and how many times this value was reached must be reported. Then for the IO consumption, the variance of the reads and writes can vary a lot and the mean is still not representative enough. We prefer to adopt a monitoring of the resources over time, that will give us all the details of the evolution of the consumptions and thus enable a deeper analysis of the cluster's resources utilization.

Monitoring and Profiling Tools. Many monitoring and profiling/tracing tools exist in the literature, but our approach needs specific requirements. First, we need to collect the jobs resource consumptions, this is a completely different process than machine monitoring. Then, we need to collect these data in a temporal way. This means that data has to be collected over time along with the jobs executions. Last, we want to collect jobs consumption data on production clusters which implies two constraints: the impact of the monitoring on the compute nodes has to be negligible; and the setup of the monitoring on the cluster has to be as simple as possible. On a running production cluster it is not acceptable to deeply modify the configuration of the nodes, nor install or update too many softwares; and the update or modification of the nodes' kernel is not possible.

Means and maximum values are not sufficient for analyzing the behavior of the applications over time, we need a more fine-grain view of their consumptions. Several systems like Ganglia[10] and Nagios[11] have been developed to monitor a system or a cluster infrastructure but are resource centric, they perform their monitoring at the machine level. Our approach needs to be job centric to enable us to extract the resource consumptions per job as proposed in [12]. Other approaches more application-oriented exist as [13] which gathers an application consumptions in an online manner taking advantage of two tools, TAU[14] as the data collector and Supermon[15] as the transport layer. Or [16] which monitors an application at runtime, with a low overhead, allowing it to study the overhead penalties incurred by Linux Systems in the execution of HPC applications. The difference between our approach and the systems mentioned above is the fact that we aim to monitor all the jobs executed in the cluster, generating a trace of the resource consumptions over time. This requires a very lightweight tool, capable of monitoring all the jobs with a low overhead and with an amount of data generated that can be easily stored and processed. These conditions were not respected by these tools.

[6] LoadLeveler: http://www-03.ibm.com/systems/software/loadleveler

Linux Kernel tools like Performance Counters[7] or CGroups[8] are not a possible option for our systems because they imply the update of the compute nodes kernel.

We also tested several event based tools like IPM[17], TAU[14], ltrace[9], but either their overhead was too high or the amount of data collected was too important (hundreds of MB for a single application run) to enable a global workload analysis. An event-driven approach would thus be too intrusive and the amount of data produced would have been too big.

The approach chosen was then a sampling monitoring, enabling us to restrict the amount of data collected and to set the precision required.
Because of the production constraints we chose to collect information about the consumptions from the */proc* subsystem (as in the OpenTSDB[10] approach). In this virtual file system (present on any Linux system), for each process is given its consumption of resources such as memory, processor, or IO on the Distributed File System (DFS). With this method, no modification nor software installation is needed on the compute nodes.

3 Resource Consumption Capture

This section presents the approach used in order to instrument the system, which allowed us to have more detailed information about jobs resources consumption during their execution. This monitoring process is divided into two parts:

- The monitoring of the jobs' processes execution in each compute node of the cluster. This is performed by a monitor daemon running on each node.
- The collection of all the traces generated by each job.

3.1 Monitor daemon

It is important to understand that we do not want to depend on the synchronization of measures. A centralized clock that forces measure times would be too intrusive for the system. Instead, every node is responsible for measuring each job every minute and the clocks between the nodes are synced (this is a common requirement in HPC). Every measure is tagged with its timestamp.

The monitor daemon, written in Perl, gathers resource consumption information for a job that is running on the machine. This information includes every process involved in the job and their resource consumption values obtained from */proc* directory. We first validated this method by making several tests with jobs whose behavior were well-known to ensure that information given by */proc* was consistent with what we expected. Collected values and trace format are described in Tables 1 and 2. In order to know which processes belong to the job, the monitor looks into the cpuset directory, which is the interface to the *cpuset*[11]

[7] https://perf.wiki.kernel.org
[8] http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt
[9] A library call tracer. http://linux.die.net/man/1/ltrace
[10] http://opentsdb.net
[11] http://www.kernel.org/doc/man-pages/online/pages/man7/cpuset.7.html

kernel mechanism to provide job isolation. Therefore with all this information collected, the monitor generates a trace file per job. The cpuset feature is supported by most of the RJMS as PBS, Loadleveler, Torque[12], Slurm[13], OAR[14], it is the most portable method for collecting the jobs processes.

We wanted the finest granularity possible without generating a big perturbation on the nodes and a minute step allowed us to have a perturbation under 1% in the worst case. It is thus a good compromise information/intrusiveness as presented in 3.2.

As these traces will be mixed with RJMS traces, a 60 seconds granularity seems reasonable as the scheduling decisions and the resources allocations are in the order of the minute.

To be as lightweight as possible, the monitoring tool does not reformat or process data during the capture. Values taken from */proc* are written directly to the log for the overhead to be in the analysis process, not during traces capture.

Trace Fields	
Name	Description
Time	Unix Time Stamp in seconds
JOB ID	Job id assigned by the batch scheduler
PID	PID of process that belongs to the job
Node ID	Provenance of the capture
Measure	Measure as presented in Table 2

Table 1: Trace Format

Values Captured	
Name	Description
command	Name of the binary executed
vmPeak	Peak virtual memory size (KB)
vmSize	Total program size (KB)
vmRss	Resident Memory (KB)
vmSwap	Size of swap usage (KB)
syscr	Number of read syscalls
syscw	Number of write syscalls
read_bytes	Bytes read from the DFS
write_bytes	Bytes written to the DFS
core	Core utilization percentage
receive	Bytes received from the network
transmit	Bytes transmitted to the network

Table 2: Data collected during a measure

3.2 Intrusiveness and Sampling Evaluation

This section presents the evaluation of our approach in terms of disruption in the system. The cost of retrieving information from the RJMS and converting it to SWF format is null as it is done post mortem. We will thus study the cost to monitor the jobs. It is very important for the validity of the data collected that the monitoring process does not interfere with the jobs themselves. It is obviously also important for the validity of the jobs results. We chose that the job overhead should be strictly less than one percent. All the tests were performed over a machine Intel Xeon E5420, with 8 cores and 24 GB of main memory. We evaluated the implementation under a 100% processor load of the machine, with one process per core and measured the monitor overhead by using the

[12] http://www.clusterresources.com/torquedocs21/3.5linuxcpusets.shtml
[13] https://computing.llnl.gov/linux/slurm/
[14] http://oar.imag.fr/sources/2.5/docs/documentation/OAR-DOCUMENTATION-ADMIN/#cpuset-feature

Sysbench multi-threaded benchmark tool. The evaluation ran 8 processes, each one checking a prime list up to 18000. Results are presented in Table 3.

Sampling Freq.	Mean	Min	Max	Overhead
No	573	573	573	0%
60 sec	573.6	573	574	0.10 %
30 sec	574.4	574	575	0.24 %
10 sec	577	576	578	0.69 %

Table 3: Overhead (in terms of perturbation in the application execution time) vs. sampling period. All values are in seconds

16 processes				
Type	Mean	Min	Max	Overhead
No monitoring	513	513	513	0 %
monitoring	514.1	514	515	0.21 %
32 processes				
Type	Mean	Min	Max	Overhead
No monitoring	512	512	512	0%
monitoring	513.8	513	516	0.35 %

Table 4: Overhead (in terms of perturbation in the application execution time) vs. number of processes monitored per job. All values are in seconds

We also evaluated the noise for more than 8 processes with a 60 seconds frequency to have an idea of the scalability of the solution. The results are shown in Table 4. The disruption caused by the monitoring was found to be almost linear regarding the number of processes to monitor in these conditions. Given that there were four times more processes than cores in the 32 processes benchmark, the result of a 0.35% overhead is acceptable.

The memory used by the daemon is very small (few MB) and it doesn't use the network during the capture. The reads and writes are a few KB every minute on the local disk and the Distributed File System is not solicited so the IO perturbation is negligible.

3.3 Collecting mechanism

Data is gathered using dedicated jobs that collect the logs when the system is underloaded. This operation does not need to be frequent; week-ends, holidays, night-times or maintenance periods can be used for this. We used a special maintenance period during holidays for this purpose. Each job requests a whole node and is in charge of gathering and compressing the trace files generated so far by the jobs executed on that machine. Then it sends them to a dedicated node in charge of storing the trace data.

3.4 Off-line data processing

Once we have collected data about jobs resources consumption we need to bind it to the RJMS log to be able to compute the resource utilization ratio. This ratio is at a given time, for a given job, the amount of resource consumed by the job over the amount of resource allocated by the system to this job. To make it simpler to use and redistributable, we first build the SWF file from the RJMS log. The SWF is an abstract view of the log, this format gives us the amount of resource allocated by the system for each job.

As measures are not guaranteed to be synchronized between nodes during the capture, a shift in the measures dates can appear. Thus we first re-sample the measures then bind them to the RJMS logs with R[15]. The whole re-sampling and binding process is available in a R script. See 5.3 for more information.

4 Experiment Environment

Data was collected from production clusters of the Ciment project[16]. This project aims at gathering several computational infrastructures to provide computing power to users in different disciplines like environment, chemistry, astrophysics, biology, health, physics. The CiGri project relies upon the OAR RJMS to submit the jobs on the clusters. All the Ciment clusters are managed by the CiGri[17] lightweight grid software that gathers clusters resources to make them available as a grid for the users.

The monitoring tool was implanted and run on two of the Ciment production clusters: *Foehn* and *Gofree*. As they come from the same grid, their respective workload should not be too different from each other. Their characteristics are detailed in table 5. A short summary of data collected is presented in Table 6.

	Foehn	**Gofree**
Brand	SGI	Dell
CPU Model	X5550	L5640
Nodes	16	28
CPU/node	2	2
Cores/cpu	4	6
Memory/node	48 GB	72 GB
Total storage	7 TB	30 TB
Network	IB DDR	IB QDR
Total Gflop/s	1367.04	3177.6
Buy date	2010-03-01	2011-01-01

Table 5: Clusters characteristics

	Foehn	**Gofree**
Capture start	2011-06-01	2011-05-24
Capture months	8	7
Log Size	2.6 GB	3.1 GB
Number of jobs	53662	20093
Besteffort jobs	50403	15596
Normal jobs	3259	4497
Active users	54	35

Table 6: Trace summary for both clusters.

One particularity of the Ciment platform is that users can submit a particular type of job: the "besteffort" jobs. These jobs are scheduled in a dedicated queue. They have a very low priority and can be preempted whenever a "normal" job arrives and needs the resource. The goal of such jobs is to maximize the cluster utilization as they are plentiful. Generally, these jobs request one core to the RJMS. These special jobs are multiparametric sequential jobs. A user using these kind of jobs has generally several instances of the same application running in different jobs with different parameters.

[15] http://www.r-project.org/
[16] CIMENT Project. https://ciment.ujf-grenoble.fr/
[17] CiGri Project: http://cigri.imag.fr/

5 Analysis

In this section we analyze the consumption of the jobs from data collected during the capture regarding their requests. This analysis is done for the following metrics: core utilization, memory utilization and IO activity on the network file system. The two clusters although belonging to the same platform are located in two different laboratories. When submitting a job, the users of the platform can both choose on which cluster the job will run or don't specify anything. In this case it is the local cluster which is selected by default. As we know, most of the users don't request specifically a cluster or tend to prefer the local cluster. Thus we analyze separately the results from the two clusters. Moreover, jobs from the **normal class** and **besteffort class** have very different patterns. Besteffort jobs are multiparametric jobs that generally request only 1 core (although we know that some besteffort jobs request several cores). They are supposed to be processor intensive with few memory usage and few IO activity. Normal jobs are generally parallel jobs requesting several nodes/cores. Their consumption patterns are not really known and are probably more varied. Hence, the following analysis of the jobs consumptions is done on the two classes separately on each cluster.

To address the analysis of the consumptions in a global way for each class we look at the resources utilization distribution. This will give us an idea of the different existing patterns.

Squashed Area Ratio. Along with the distribution of the utilizations we will also consider the impact of a particular phenomenon into the global system activity. We present thus the Squashed Area metric (SA) that represents the jobs execution activity. [18] defines SA as the total system's resource consumptions of a set of jobs, computed by:
$SA = \sum_{j \in jobs} allocated_cores_j \times run_time_j$.

Thus, for a job or a set of jobs, we look at the SA ratio of this set regarding the workload SA of its class (i.e. the proportion of this set area over the total class area). This enables us to determine if a particular phenomenon is significant enough regarding the rest of the workload.

Core. In data collected we have for each job j, for each core c allocated to j, the proportion of core consumed by the processes running on c and belonging to j along its execution. We denote this value: $core_consumption_j^c(t_i)$. With t_i being the date of the measure i. Thus, we can compute for each measure date t_i the total amount of core consumed by a job with:
$core_consumption_j(t_i) = \sum_{c \in allocated_cores_j} core_consumption_j^c(t_i)$.
Then, with n being the number of consumption measures taken for j, we can

compute per job its core utilization mean by:

$$core_utilization_mean_j = \frac{\sum_{i \in [1,n]} core_consumption_j(t_i)}{allocated_cores_j \times n}.$$

In the following analysis we consider the distribution of these values to identify different utilization patterns.

Memory. The OAR version managing Foehn and Gofree clusters doesn't provide a memory isolation of the jobs on the nodes. This feature in OAR is only available with the Cgroup feature of the kernel enabled which is not the case. Thus it is interesting to look whether jobs use a lot of memory or not. Indeed, a job which uses intensively the memory could disturb other jobs sharing the same node.

We introduce the *theoretical_memory_per_core* value which is equal to a node memory over the number of cores on the node. As each cluster has an homogeneous architecture, this value is the same for all the nodes in a cluster. Foehn and Gofree doesn't have the same number of cores and memory per node but their *theoretical_memory_per_core* is the same and is equal to 6GB. We consider thus that a job which consumes less than this value per allocated core is not disturbing for the other jobs.

Then, for a given job we know its theoretical maximum memory. This value is equal to $theoretical_memory_per_core \times allocated_cores_j$. Thus we know that jobs that consume more than 100% of this value are potentially disturbing other jobs by using too much memory on at least one node.

For the analysis of the memory utilization we will look at two sub-metrics:

- The mean memory used by a job (computed by the same method than the core utilization mean) vs. its theoretical maximum memory. This tells us how the job behaves on average regarding its theoretical maximum memory.
- The maximum memory used on the cores allocated to this job. This will tell us if the job used more than the *theoretical_memory_per_core* on one of its allocated cores or not.

File System IO. As the Distributed File System (DFS) is a central component of a cluster we are also interested into its usage patterns. On Gofree no user has complained about slow IO on the DFS. This is not the case for Foehn where users have reported several IO problems. Unfortunately we could only monitor the IO utilization on the Distributed File System on the Gofree cluster and this for only two months and a half. Gofree DFS consists in an NFS server that serves the users' home directories through the GigaBit Ethernet network. We tested its capabilities in terms of bandwidth with the IOR[18] benchmark[19]. We used IOR in a single file/single client mode with file sizes of 64MB and blocks of 4KB. 4KB is the default block size. 64MB and 64KB are the most frequent write sizes in

[18] http://sourceforge.net/projects/ior-sio/

Gofree (see Figure 15). We chose 64MB instead of 64KB because this last value is too small (only 16 physical writes) to have a correct idea of the bandwidth. We repeated the test 10000 times and got a max speed of 103MB/s for the writes and 4030MB/s for the reads. The mean write speed was 98MB/s with a standard deviation of 5. Several other tests with higher values of file sizes gave us similar results. For concurrent sequential tests IOR showed that the bandwidth was more or less fairly divided between the different writing processes. For parallel tests (with MPIIO API, one file per process, up to 48 processes) IOR showed a maximum value close to 96MB/s which is a little less than with the single process mode but still in the same bandwidth range than the mean speed of sequential writes. This gives us an approximative idea of the global bandwidth of Gofree DFS.

The read bandwidth is quite high (close to a local file system bandwidth) and generally the type of IO which are problematic are the writes, thus we will focus on the analysis of the writes patterns.

5.1 Foehn Cluster

Normal Class Jobs. Figure 1 presents the distribution of the core utilization means for normal class jobs on Foehn cluster. We observe a peak in the distribution corresponding to a core utilization near 100%. An other peak, less important corresponds to a low core utilization (between 0% and 25%). However when looking at the SA ratio of these two peaks the tendency reverts. The SA ratio of the high utilization peak is only 12.7% of the total workload SA and the SA ratio of the low utilization peak is of 53.7%. For these jobs, very few of them consume a lot of memory (only 22 jobs). Among them only 5 consume more than their theoretical maximum memory. For these memory intensive jobs, the maximum memory used per core is 125% of the *theoretical_memory_per_core*. Figure 2 presents the distribution of the mean memory used per job. We can observe that memory intensive jobs are very rare on Foehn but exist.

Figures 3 and 4 present the maximum memory used on a core per job. We split the representation of the distribution in two groups for more clarity: up to 100% and more than 100%.

There are 31 jobs whose mean memory usage is over 100%. Unsurprisingly, jobs that have a mean memory usage over 100% are the same than the jobs that have a peak memory usage on an allocated core over 100%. These jobs have a peak value grouped around 140%. Only 1 job has a very high peak memory usage of 480%, it is a job that requested 2 cores and lasted for 18 minutes then was canceled by its user.

In Foehn, we observe that many jobs are using the cores computing power very efficiently and don't use a lot of memory. These jobs seem to be cpu bound. However their weight in the workload is counterbalanced by larger and shorter jobs (larger number of resources but with a runtime being a little shorter on average). Regarding the memory usage distribution, we can think that these jobs suffered from something else than memory problems. But when we look at the number of cores allocated to the jobs of the two groups (jobs with average

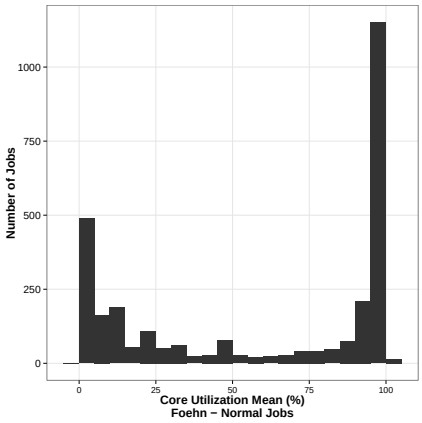

Fig. 1: Distribution of the jobs' core utilization means.

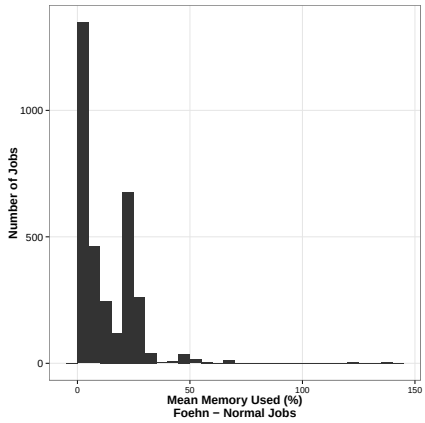

Fig. 2: Distribution of the mean memory used by job.

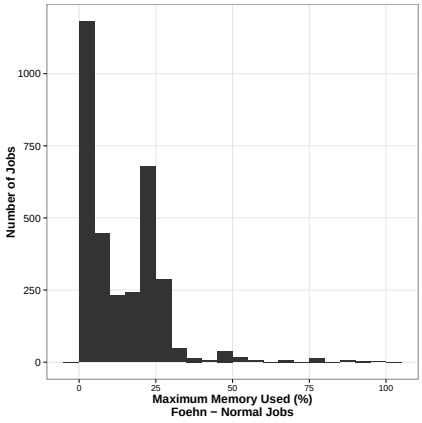

Fig. 3: Distribution of the maximum memory used per job by an allocated core to this job, values$\leqslant$100%.

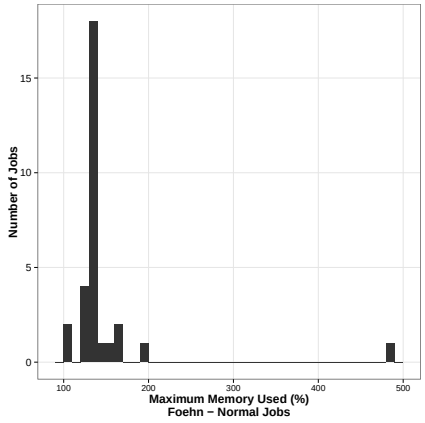

Fig. 4: Distribution of the maximum memory used per job by an allocated core to this job, values>100%.

core utilization under 25% and jobs with average core utilization near 100%) we remark something very particular. The mean number of allocated cores is 4 times bigger for the jobs that have a core utilization in the lower peak. This can be an IO scalability issue with jobs waiting on IO, as we know that Foehn DFS has bandwidth problems, this having been reported several times by the users. However this can also come from an over-reservation of cores by the users. Sadly, we currently miss information to explain the reason of this phenomenon.

Besteffort Class Jobs. Figure 5 presents the distribution of core utilization for besteffort jobs on Foehn cluster. We observe two peaks, one corresponding to a core utilization mean around 25%, the other around 100%. For the lower

utilization peak (from 0 to 25%), its SA ratio is 55.5%. The SA ratio for the jobs exactly in the 25% peak is 42.4%. For the jobs involved in the higher distribution peak, even though there are fewer, their SA ratio is about the same (40.8%). 99.2% of these jobs reserve only 1 core. These jobs never use more than the theoretical memory per core. 99.5% of them actually use less than 1/3 of this theoretical memory per core.

Only 6 users are involved in the low core utilization peak, but 5 of them are also involved in the high utilization peak. The user only present in the low utilization peak accounts for 41.9% of the besteffort SA and has a core utilization less than 25% in 99.4% of the cases. He reserves 4 cores in all his besteffort jobs. In almost all the cases his processes consume less than 310MB and never higher than 620MB.

After contacting the user, he explained that he noticed that if two of his jobs were running on the same node, the performances of the two jobs were very bad. After investigating the cause of this it was found that the application was memory intensive in terms of bandwidth. The maximum memory used was small compared to the amount of memory available on the node but when two jobs were accessing the memory at same time, there was a bottleneck in the access to the memory. The solution adopted by the user to avoid this performance loss was to reserve the whole socket (corresponding thus to 4 cores) even though the application only used 1 core. As Foehn nodes are NUMA, reserving the whole socket enabled the jobs to have their own memory slot and thereby not being disturbed. The problem here was not a misconfiguration of the job or a bad reservation request but a lack in the RJMS constraint description that forced the user to over-reserve.

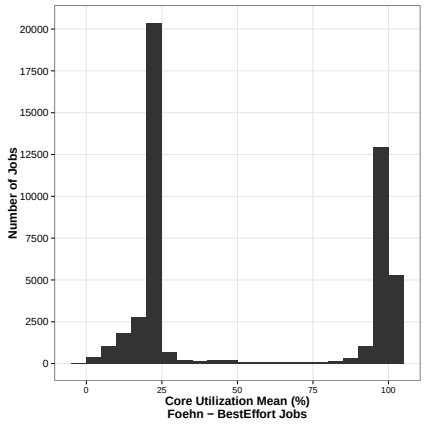

Fig. 5: Distribution of the jobs' core utilization means.

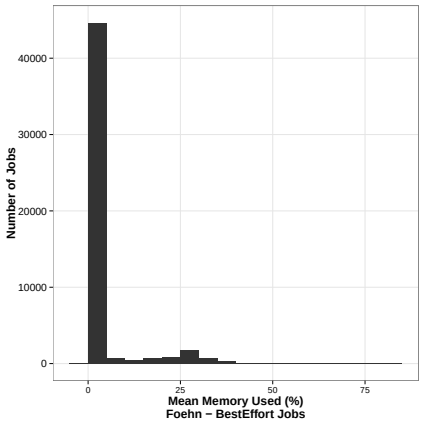

Fig. 6: Distribution of the mean memory used by job.

Figure 6 shows that besteffort jobs don't consume a lot of memory on average. Only 6 jobs (not represented on the figure for clarity) have a mean memory

utilization greater than their theoretical available memory. Their mean memory use is between 7 and 8 GB.

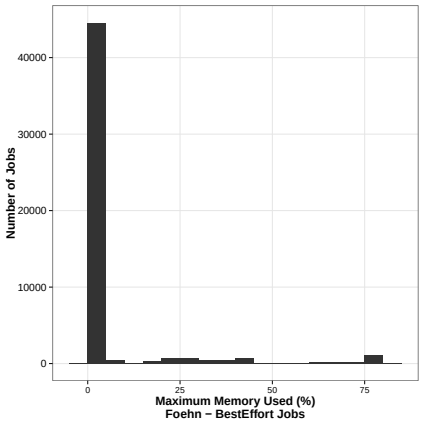

Fig. 7: Distribution of the maximum memory used per job by an allocated core to this job, values⩽100%.

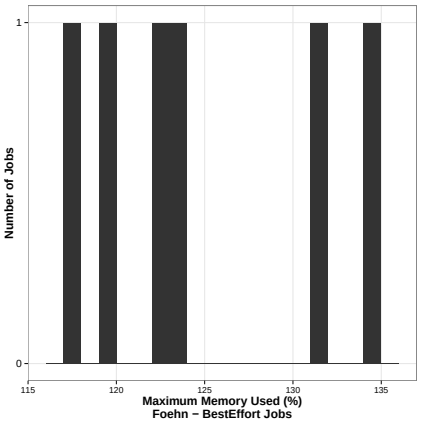

Fig. 8: Distribution of the maximum memory used per job by an allocated core to this job, values>100%.

In Figures 7 and 8 we observe that besteffort jobs don't have big memory peaks. Only 6 jobs have peaks between 115 and 135%. these jobs are all 1 core jobs and five of them belong to the same user.

Except the singularity of the user reserving one entire socket and few jobs with memory usage peaks, besteffort jobs on Foehn tend to be cpu bound.

5.2 Gofree Cluster

Normal Class Jobs. Figure 9 shows the distribution of the core utilization means for jobs in the normal class in Gofree. We observe a very particular distribution with four peaks around 0, 1/4, 1/2 and 1 of core utilization mean. The most surprising is when we look at the number of cores allocated to these jobs vs their core utilization. The higher peak has a core allocation of 1 node in 50% of the cases (with a maximum of 1 node). The peak around 1/2 has a core allocation of 2 nodes in 85% of the cases (with a maximum of 2 nodes). The peak around 1/4 has a core allocation of 3 nodes in 48% of the cases and 4 nodes in 35% of the cases (with a maximum of 4 nodes). The jobs below 50% of core utilization are not an isolated phenomenon, 2/3 of the users are involved in these jobs and their presence is distributed all along the trace. These users are also involved in the high utilization peak. The memory used for these jobs is low (less than 1/3 of theoretical memory) for 89% of them. Only 7 jobs use up to 100% of their theoretical available memory. Figures 10 and 11 show us a memory utilization quite low for most of the jobs of this class.

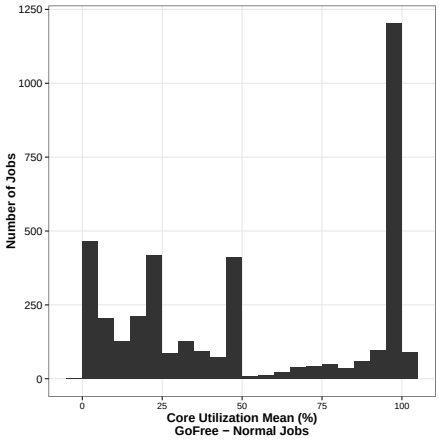

Fig. 9: Distribution of the jobs' core utilization means.

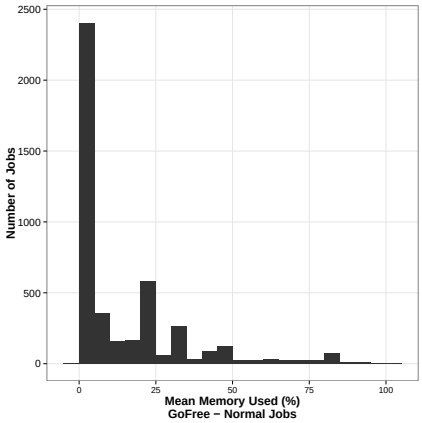

Fig. 10: Distribution of the mean memory used by job.

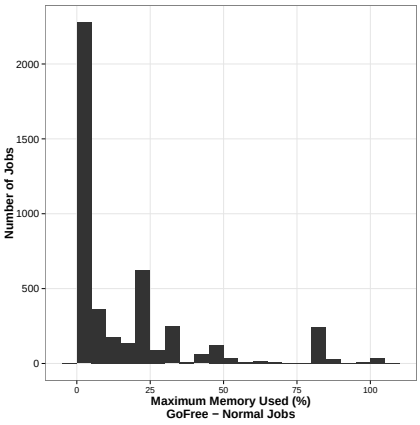

Fig. 11: Distribution of the maximum memory used per job by an allocated core to this job.

This behavior of core utilization vs core allocation is very noteworthy and we suspect a scalability problem although we can't say where is the bottleneck (results of IO consumption on data collected during the IO capture period showed the DFS is usually not very stressed, see 5.2). However, as for Foehn, this can also come from an over-reservation of cores by the users. These two phenomena of low core utilization on both clusters will need further investigations and the jobs' knowledge of the users involved.

Besteffort Class Jobs. Figure 12 shows that besteffort jobs in Gofree have a high core utilization. Jobs below 75% of core utilization represent 0.37% of the normal jobs SA and jobs below 90% account for 7% of the class' SA. Core allocation for besteffort jobs in 85% of the cases is 1 with other values being 4

and 6 (1 cpu socket). Figures 13 and 14 show us a low memory utilization on average and on maximum. Regarding these results we can say that besteffort jobs in Gofree are cpu bound.

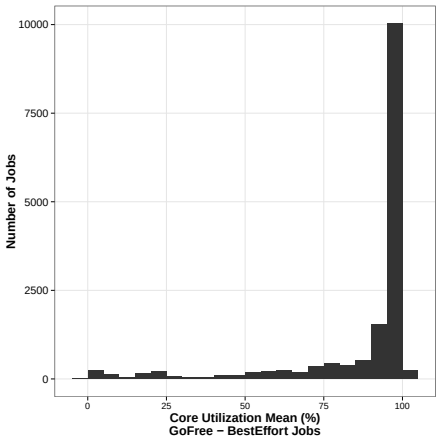

Fig. 12: Distribution of the jobs' core utilization means.

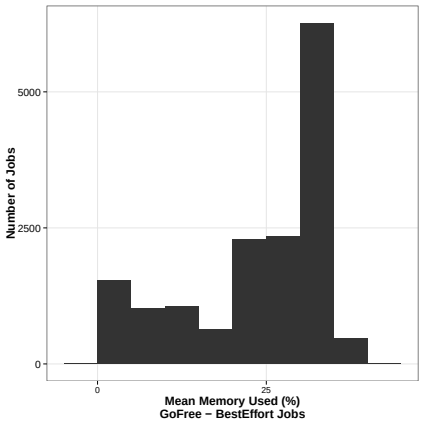

Fig. 13: Distribution of the mean memory used by job.

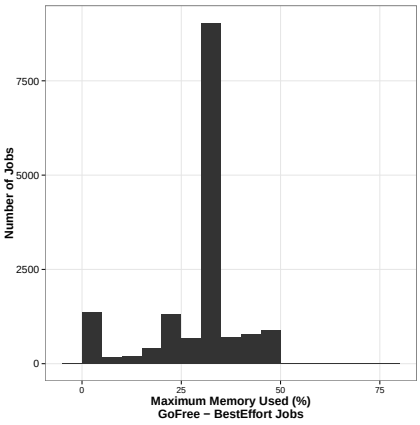

Fig. 14: Distribution of the maximum memory used per job by an allocated core to this job.

Global IO Activity. Figure 15 shows the distribution of the size of the files written by the jobs. We can observe a big peak at 64KB. This comes from many besteffort jobs that write small files. Generally, besteffort jobs tend to write either files of 64KB or 32MB. Normal jobs tend to write files of 4KB and 16MB.

Figure 16 presents the distribution of the aggregated writes by the jobs on the File System. We can observe that the most frequent sizes the DFS has to write in a minute is either small (between 4KB and 16KB) or medium (between

8MB and 64MB). It seems that the DFS is not really overloaded. Very few values are high (up to 6GB in a minute).

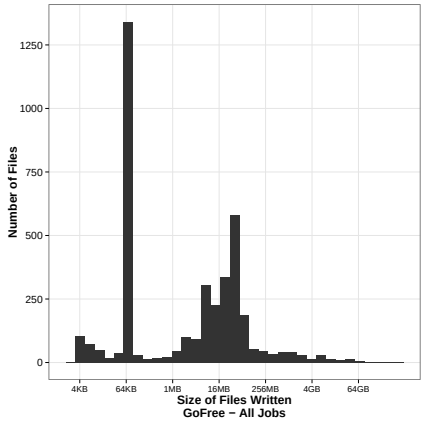

Fig. 15: Size of Files Writes on the DFS on Gofree cluster for all jobs.

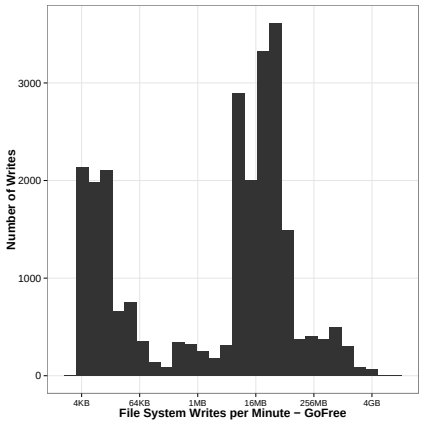

Fig. 16: Distribution of the aggregated File System Writes sizes (per Minute) on Gofree cluster.

In order to see how the load is distributed along the time we plot Figures 17 and 18. Figure 17 presents the aggregated writes per day on the DFS. We can observe that the daily IO write activity is very irregular with intense periods and an empty period. The period with no IO activity occurred during holidays, there was almost no job submitted during this time.

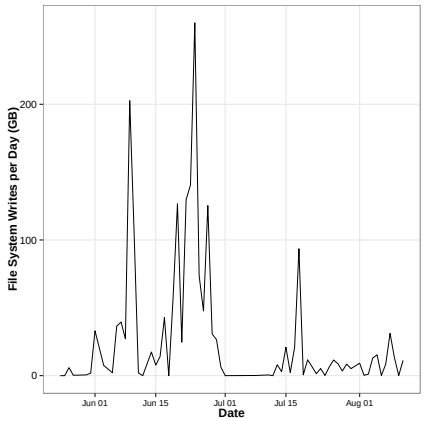

Fig. 17: Daily aggregated writes on the DFS.

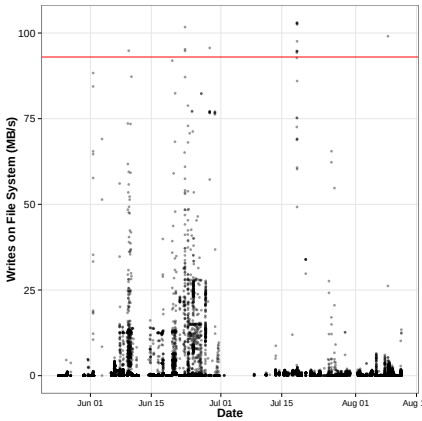

Fig. 18: DFS load (write speed) on Gofree cluster.

Figure 18 plots the write load in terms of speed on the Gofree DFS server. In this figure, we isolate some remarkable values above the vertical line at 93MB/s. The value of 93MB/s has been chosen because it reflects the speed where the DFS might start to be overloaded. It corresponds to the mean write speed minus the standard deviation given by our bandwidth tests with IOR. We refer at the area above this line as the DFS hot zone.

We observe five ranges of dates where the DFS is in the hot zone. These dates are respectively the June 9, June 22, June 28, July 18 and August 8. The most remarkable period is July 18 where in a period of two hours the DFS reached the hot zone 10 times. When looking at the jobs involved in this activity we see that two jobs (belonging to two different users) where competing for IO on the DFS. The jobs where using 4 and 12 nodes. They didn't use much memory and their mean core utilization was quite low. The job using 4 nodes was canceled by the user before its end. This particular scenario let's us think that this job that was canceled was suffering for slow IO and thus killed by its owner. The scenario is about the same during June 22 where four jobs where competing on IO. One ended, canceled by the user and one had an IO write pattern alternating big writes with periods of inactivity (few memory used, no core activity) probably being blocked on IO. However this event lasted for only 10 minutes. The other periods are less remarkable as the presence of the DFS in the hot zone was brief (only one peak) with few jobs involved in IO activity.

5.3 Reproducibility

All the tools, traces and R-Sweave documents used to process this analysis are available on a Git repository[19] for the sake of reproducibility and sharing.

6 Discussion and Perspectives

Monitoring the jobs resource consumptions and linking it to their resource allocation enabled us to detect some particular behaviors on the clusters. Moreover, the monitoring over time allowed us to have a more precise view of the resources utilization, particularly it enabled us to reconstruct the Distributed File System (DFS) write load along time. The study of the utilization for the different resource types gave us a better comprehension of our users' usage of the clusters. We are now able to propose some enhancements in our future scheduling strategies based on the observed usage of the system.

The problem of memory bandwidth risen in the analysis is very interesting because we can encounter the same lack of constraint description for memory, IO or network in most current RJMS. As the Squashed Area ratio of the user involved in this phenomenon weights a lot regarding the rest of the workload, the core computing power loss is important. Here it is not a resource quantity problem, but a problem concerning the bandwidth to access the resource. We

[19] git clone https://forge.imag.fr/anonscm/git/evalys-tools/evalys-tools.git

cannot try strategies like scheduling other besteffort jobs on the same socket since we don't know their bandwidth usage patterns. A possible solution would be to define memory, IO and network bandwidths as resources in the RJMS in the same way cores are defined as resources. Thus we enable the user to reserve not only a set of cores but an amount of bandwidth on the node depending his/her need. However, this kind of guarantee is not possible nowadays.

It is also noteworthy that besteffort jobs on both clusters are core efficient with a low memory usage. This will be used as a scheduling tweak; besteffort jobs, whenever it is possible, should be scheduled on the same nodes. As we now are sure that they will not disturb with one another, we will pack these jobs on the same set of nodes to improve their efficiency. Besteffort jobs are killed whenever a normal job needs the resource they are running on. By packing them we reduce the fragmentation and thus the probability for them to be killed.

The problem of the DFS overload is also interesting. We observed that generally the DFS is not too loaded but when it happens and there are jobs competing for IO it ends with the cancellation of one of these IO intensive jobs. Giving the users the possibility to tag their jobs (e.g. as IO intensive) at the submission to the RJMS might prevent such situations, the RJMS will simply not schedule two IO intensive jobs at the same time.

Monitoring jobs resource consumptions revealed problems that were not visible with the sole System Utilization metric. Coupling jobs consumptions and RJMS logs enabled us to exhibit and quantify significant particular cases. On mid-sized computing centers, the study of the jobs resource utilization will become a necessary way to a deeper understanding of the users needs and their jobs consumption patterns. Particular patterns discovered by the analysis of the jobs resource utilization will lead to dedicated system setup and optimizations to improve both users and administrators satisfaction.

References

1. Ernemann, C., Song, B., Yahyapour, R.: Scaling of workload traces. In Feitelson, D., Rudolph, L., Schwiegelshohn, U., eds.: Job Scheduling Strategies for Parallel Processing. Volume 2862 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2003) 166–182
2. Lublin, U., Feitelson, D.G.: The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. Journal of Parallel and Distributed Computing **63** (2001) 2003
3. Feitelson, D.: Workload modeling for performance evaluation. In Calzarossa, M., Tucci, S., eds.: Performance Evaluation of Complex Systems: Techniques and Tools. Volume 2459 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2002) 114–141
4. Rudolph, L., Smith, P.: Valuation of ultra-scale computing systems. In Feitelson, D., Rudolph, L., eds.: Job Scheduling Strategies for Parallel Processing. Volume 1911 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2000) 39–55

5. Zhang, Y., Sivasubramaniam, A., Moreira, J., Franke, H.: Impact of workload and system parameters on next generation cluster scheduling mechanisms. Volume 12. (2001) 967–985
6. Chapin, S.J., Cirne, W., Feitelson, D.G., Jones, J.P., Leutenegger, S.T., Schwiegelshohn, U., Smith, W., Talby, D.: Benchmarks and standards for the evaluation of parallel job schedulers. In Feitelson, D.G., Rudolph, L., eds.: Job Scheduling Strategies for Parallel Processing. Springer-Verlag (1999) 67–90 Lect. Notes Comput. Sci. vol. 1659.
7. Jain, R.: The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation, and modeling. Wiley (1991)
8. Yoo, A., Jette, M., Grondona, M.: Slurm: Simple linux utility for resource management. In Feitelson, D., Rudolph, L., Schwiegelshohn, U., eds.: Job Scheduling Strategies for Parallel Processing. Volume 2862 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2003) 44–60
9. Capit, N., Costa, G.D., Georgiou, Y., Huard, G., Martin, C., Mounie, G., Neyron, P., Richard, O.: A batch scheduler with high level components. Cluster Computing and the Grid (2005) 776–783
10. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: Design, implementation and experience (2004)
11. Imamagic, E., Dobrenic, D.: Grid infrastructure monitoring system based on nagios. In: Proceedings of the 2007 workshop on Grid monitoring. GMW '07, New York, NY, USA, ACM (2007) 23–28
12. Curry, R., Simmonds, R.: Job centric cluster monitoring. In: Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on. Volume 1. (0-0 2006) 8 pp.
13. Nataraj, A., Sottile, M., Morris, A., Malony, A., Shende, S.: Tauoversupermon: Low-overhead online parallel performance monitoring (2007)
14. Shende, S.S., Malony, A.D.: The tau parallel performance system. The International Journal of High Performance Computing Applications **20** (2006) 287–331
15. Sottile, M.J., Minnich, R.G.: Supermon: A high-speed cluster monitoring system. In: Proceedings of the IEEE International Conference on Cluster Computing. CLUSTER '02, Washington, DC, USA, IEEE Computer Society (2002)
16. Sharma, S., Bridges, P.G., Maccabe, A.B.: A framework for analyzing linux system overheads on hpc applications. In: Proceedings of the 2005 Los Alamos Computer Science Institute Symposium. (October 2005)
17. Fuerlinger, K., Wright, N.J., Skinner, D.: Effective performance measurement at petascale using ipm. In: Proceedings of The Sixteenth IEEE International Conference on Parallel and Distributed Systems (ICPADS 2010), Shanghai, China (December 2010)
18. Song, B., Ernemann, C., Yahyapour, R.: Parallel computer workload modeling with markov chains. In Feitelson, D., Rudolph, L., Schwiegelshohn, U., eds.: Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2005)
19. Shan, H., Antypas, K., Shalf, J.: Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. SC '08, Piscataway, NJ, USA, IEEE Press (2008) 42:1–42:12