Evaluating scalability and efficiency of the Resource and Job Management System on large HPC Clusters

Yiannis Georgiou[1], Matthieu Hautreux[2]

[1]BULL S.A.S
Yiannis.Georgiou@bull.fr
[2] CEA-DAM
Matthieu.Hautreux@cea.fr

Abstract. The Resource and Job Management System (RJMS) is the middleware in charge of delivering computing power to applications in HPC systems. The increasing number of computational resources in modern supercomputers brings new levels of parallelism and complexity. To maximize the global throughput while ensuring good efficiency of applications, RJMS must deal with issues like manageability, scalability and network topology awareness. This paper is focused on the evaluation of the so-called RJMS SLURM regarding these issues. It presents studies performed in order to evaluate, adapt and prepare the configuration of the RJMS to efficiently manage two Bull petaflop supercomputers installed at CEA, Tera-100 and Curie. The studies evaluate the capability of SLURM to manage large numbers of compute resources and jobs as well as to provide an optimal placement of jobs on clusters using a tree interconnect topology. Experiments presented in this paper are conducted using both real-scale and emulated supercomputers using synthetic workloads. The synthetic workloads are derived from the ESP benchmark and adapted to the evaluation of the RJMS internals. Emulations of larger supercomputers are performed to assess the scalability and the direct eligibility of SLURM to manage larger systems.

1 Introduction

The advent of multicore architectures and the evolution of multi-level/multi-topology interconnect networks has introduced new complexities in the architecture as well as extra levels of hierarchies. The continuous growth of cluster's sizes and computing power still follows Moore's law and we witness the deployment of larger petascale supercomputing clusters [1]. Furthermore, the continuous increasing needs for computing power by applications along with their parallel intensive nature (MPI, OpenMP, hybrid,...) made them more sensitive to communication efficiency, demanding an optimal placement upon the network and certain quality of services.

The work of a Resource and Job Management System (RJMS) is to distribute computing power to user jobs within a parallel computing infrastructure. Its goal is to satisfy users demands for computation and achieve a good performance in overall system's utilization by efficiently assigning jobs to resources. This assignment involves three principal abstraction layers: the declaration of a job where the demand of resources and job characteristics take place, the scheduling of the jobs upon the resources given their organization and the launching of job instances upon the computation resources along with the job's control of execution.

The efficient assignment of large number of resources to an evenly large number of users jobs arises issues like job launchers and scheduling scalability. The increase of network diameter and the network contention problem that can be observed in such large network sharing scenarios demand a certain consideration so as to favor the placement of jobs upon groups of nodes which could provide optimal communication characteristics. Since the RJMS has a constant knowledge of both workloads and computational resources, it is responsible to provide techniques for the efficient placement of jobs upon the network.

BULL and CEA-DAM have been collaborating for the design, construction or installation of 2 petascale cluster systems deployed in Europe. The first one, "Tera100" [1], with a theoretical computing power of 1.25 petaflops, is in production since November 2010 and represents Europe's most powerful system and 9Th more powerful in the world, according to November's 2011 top500 list [1]. The second, the French PRACE Tier0 system, "Curie" [2], with a theoretical computing power of 1.6 petaflops is currently under deployment and planned to be in production on March 2012. The Resource and Job Management System installed on both systems is SLURM [2] which is an open-source RJMS specifically designed for the scalability requirements of state-of-the-art supercomputers.

The goal of this paper is to evaluate SLURM's scalability and jobs placement efficiency in terms of network topology upon large HPC clusters. This study was initially motivated by the need to confirm that SLURM could guarantee the efficient assignment of resources upon user jobs, under various realistic scenarios, before the real deployment of the above petascale systems. We wanted to reveal possible weaknesses of the software before we come up with them into real life. Once the desired performance goals were attained, the studies continued beyond the existing scales to foresee the efficiency of SLURM on even larger clusters.

Inevitably the research on those systems implicate various procedures and internal mechanisms making their behavior complicated and difficult to model and study. Every different mechanism depends on a large number of parameters that may present inter-dependencies. Thus, it is important to be able to study the system as a whole under real life conditions. Even if simulation can provide important initial insights, the need for real-scale experimentation seems necessary for the study and evaluation of all internal functions as one complete system. On the other hand, real production systems are continuously overbooked for scientific applications execution and are not easily available for this type of experiments in full scale. Hence, emulation seems to be the best solution for large-scale experimentations.

In this paper we present a real-scale and emulated scale experimental methodology based upon controlled submission of synthetic workloads. The synthetic workloads are derived from the ESP model [3, 4] which is a known benchmark used for the evaluation of launching and scheduling parameters of Resource and Job Management Systems. In this study the default ESP workload was modified in order to adapt to the characteristics of real large cluster usage. Hence, the experimental methodology makes use of two derived variations of ESP which are introduced in this paper: the Light-ESP and the Parallel Light-ESP. The experiments took place upon a subset of Tera-100 cluster during maintenance periods and upon an internal BULL cluster dedicated for research and development.

The remainder of this article is presented as follows: The next section provides the Background and Related Work upon RJMS and performance evaluation for these type of systems. Section 3 describes the experimental methodology that has been adopted and used for the evaluation of the RJMS. Section 4 provides the main part of this study where we present and discuss our evaluation results upon the scalability and efficiency of SLURM RJMS. Finally the last section presents the conclusions along with current work and perspectives.

2 Background and Related Work

Since the beginning of the first Resource and Job Management Systems back in the 80s, different software packages have been proposed in the area to serve the needs of the first HPC Clusters. Nowadays, various software exist either as evolutions of some older software (like PBSPro or LSF) or with new designs (like OAR and SLURM). Commercial systems like LSF [5], LoadLeveler [6], PBSPro [7] and Moab [8] generally support a large number of architecture platforms and operating systems, provide

[1] http://www.hpcwire.com/hpcwire/2010-05-27/tera_100_europes_most_powerful_supercomputer_powers_up.html

[2] http://www.prace-ri.eu/CURIE-Grand-Opening-on-March-1st-2012?lang=en

highly developed graphic interface for visualization, monitoring and transparency of usage along with a good support for interfacing standards like parallel libraries, Grids and Clouds. On the other hand their open-source alternatives like Condor [9], OAR [10], SLURM [2], GridEngine [11], Torque [12] and Maui [13] provide more innovation and a certain flexibility when compared to the commercial solutions.

2.1 Evaluating the internals of Resource and Job Management Systems

Numerous studies have been made to evaluate and compare different Resource and Job Management Systems [14], [15].

One of the first studies of performance evaluations for RJMS was presented in [16]. In this study Tarek et al. have constructed a suite of tests consisted by a set of 36 benchmarks belonging to known classes of benchmarks (NSA HPC, NAS Parallel, UPC and Cryptographic). They have divided the benchmarks into four sets comprising short/medium/long and I/O job lists and have measured average throughput, average turn-around time, average response time and utilization for 4 known Resource and Job Management Systems: LSF, Codine, PBS and Condor.

Another empirical study [10] measured throughput with submission of large number of jobs and efficiency of the launcher and scheduler by using a specific ESP benchmark. The study compared the performance of 4 known RJMS: OAR, Torque, Torque+Maui and SGE. ESP benchmark has similarities with the previously described suite of tests [16]. However one of its advantages is that it can result in a specific metric that reflects the performance of the RJMS under the specific workload and the selected scheduling parameters. The approach used in this work is an extension of the methodology used for the performance evaluation in [10], since we propose a similar method using variations of the ESP benchmark workload model.

Performance evaluation is effectuated by having the system's scheduler schedule a sequence of jobs. This sequence is the actual workload that will be injected to the system. Different researches [17], [18], [19], [20] has been effectuated upon workload characterization and modeling of parallel computing systems. In order to model and log a workload of a parallel system, Chapin et al. [19] have defined the *standard workload format* (swf) which provides a standardized format for describing an execution of a sequence of jobs. In the internals of our experimentation methodology we are based on the swf format for the workload collection mechanisms and make use of particularly developed tools for swf workload treatment and graphic representations.

Based on previous work [20], [21] it seems that to give precise, complete and valuable results the studies should include the replay of both modeled and real workload traces. In our studies we have considered only the usage of synthetic workloads for the moment but once the deployment of "Curie" platform has been made we will collect the real workload traces of the petaflop system and either replay directly parts of them or try to extract the most interesting information out of them to construct new models which will be closer to the real usage of this system. Nevertheless, we believe that in those cases the reproduction of experiments and the variations of different factors results into reliable observations.

2.2 Network Topology aware Placement

Depending on the communication pattern of the application, and the way processes are mapped onto the network, severe delays may appear due to network contention, delays that result in longer execution times. Nodes that are connected upon the same switch will result in better parallel computing performance than nodes that are connected on different switches. Mapping of tasks in a parallel application to the physical processors on a machine, based on the communication topology can lead to performance improvements [22]. Different solutions exist to deal with those issues on the resource management level.

We are especially interested on fat-tree network topologies which are structures with processing nodes at the leaves and switches at the intermediate nodes [23]. As we go up the tree from the leaves, the

available bandwidth on the links increases, making the links ”fatter”. Network topology characteristics can be taken into account by the scheduler [24] so as to favor the choice of group of nodes that are placed on the same network level, connected under the same network switch or even placed close to each other so as to avoid long distance communications.

This kind of feature becomes indispensable in the case of platforms which are constructed upon pruned fat-tree networks [25] where no direct communication exist between all the nodes. This reduces the number of fast communication group of nodes to only those connected under the same switch. Hence, the efficient network placement is an important capability of a Resource and Job Management System that may improve the application performance, decrease their execution time which may eventually result into smaller waiting times for the other jobs in the queue and an overall amelioration of the system utilization. SLURM provides topology aware placement techniques, treated as an extra scheduling parameter and based upon best-fit algorithms of resources selection. As far as our knowledge, even if other RJMS provide techniques for topology aware placement, SLURM is the only RJMS that proposes best-fit selection of resources according to the network design. In this study we make an in-depth performance evaluation of the topology aware scheduling efficiency of SLURM.

2.3 Scalability

Scalability is one of the major challenges of large HPC systems. It is related with different issues on various levels of the software stack. In the context of Resource and Job Management Systems we are mainly interested into scalability in terms of job launching, scheduling and system responsiveness and efficiency when increasing the number of submitted jobs and the systems scale. Previous related work in the field [26, 27] have produced a flexible, lightweight Resource Management called STORM which provided very good scalability in terms of job launching and process scheduling and produced experiments showed very good results in comparison with SLURM or other RJMS. However, on the one hand the particular software was more a research tool and not a production RJMS and it did not provide any backfilling or more elaborate scheduling techniques like network topology consideration.

Another scalable lightweight software used for fast deployment of jobs is Falkon [28]. This system is used as a meta-scheduler by connecting directly upon the RJMS and taking control over the resources by simplifying the procedure of jobs deployment. Hence again in this case the focus is centered just on the fast deployment of jobs without any concern about optimized task placement or prioritization between jobs. A group of computer scientists in CERN have performed scalability experiments of LSF scheduler[3] where they experiment with large scale deployment of LSF scheduler (up to 16000 nodes) using virtual machines. They resulted into good scaling performance of the LSF scheduler but there was no reported analysis of the experiments in order to be able to reproduce them.

In our context of large scale HPC systems we are interested into scalability but also keeping the elaborate scheduling capabilities of the RJMS in order to guarantee a certain quality of service to the users. Hence in this study we are trying to explore the scalability of SLURM by keeping into consideration that efficiency and user quality of services should not be disregarded while we increase in larger scales. As far as our knowledge no other studies have been found in the literature that evaluate the scalability and topology aware scheduling efficiency of Resource and Job Management Systems upon large HPC clusters.

[3] http://indico.cern.ch/contributionDisplay.py?contribId=7&sessionId=7&confId=92498

3 Experimentation Methodology for RJMS scalability and efficiency

3.1 Evaluation based upon Synthetic Workloads

To evaluate the scheduling performance of Resource and Job Management Systems, we have adopted the Effective System Performance (ESP) model [3, 4]. The initial version of this benchmark not only provides a synthetic workload but proposes a specific type of parallel application that can be executed to occupy resources as simply as possible. In our case we just make use of the ESP synthetic workload model and execute simple sleep jobs in place of the proposed parallel application. Indeed, in the context of our experiments which is to evaluate the efficiency of the scheduler and not the behavior of the clusters' runtime environment, the choice of simple sleep jobs is enough.

The ESP [4] test was designed to provide a quantitative evaluation of launching and scheduling parameters of a Resource and Job Management System. It is a composite measure that can evaluate the system via a single metric, which is the smallest elapsed execution time of a representative workload. In ESP, there are 230 jobs derived from a list of 14 job types, as shown in detail in table 1, which can be adjusted in a different proportional job mix . The test is stabilized to the number of cores by scaling the size of each job with the entire system size. Table 1 shows the fraction of each class's job size along with the the number of jobs and the respective duration.

Benchmarks	Normal-ESP	Light-ESP	Parallel Light-ESP
Job Type	Fraction of job size relative to system size (job size for cluster of 80640 cores) Number of Jobs / Run Time (sec)		
A	0.03125 (2520) / 75 / 267s	0.03125 (2520) / 75 / 22s	0.003125 (252) / 750 / 22s
B	0.06250 (5040) / 9 / 322s	0.06250 (5040) / 9 / 27s	0.00625 (504) / 90 / 27s
C	0.50000 (40320) / 3 / 534s	0.50000 (40320) / 3 / 45s	0.05000 (4032) / 30 / 45s
D	0.25000 (20160) / 3 / 616s	0.25000 (20160) / 3 / 51s	0.02500 (2016) / 30 / 51s
E	0.50000 (40320) / 3 / 315s	0.50000 (40320) / 3 / 26s	0.05000 (4032) / 30 / 26s
F	0.06250 (5040) / 9 / 1846s	0.06250 (5040) / 9 / 154s	0.00625 (504) / 90 / 154s
G	0.12500 (10080) / 6 / 1334s	0.12500 (10080) / 6 / 111s	0.01250 (1008) / 60 / 111s
H	0.15820 (12757) / 6 / 1067s	0.15820 (12757) / 6 / 89s	0.01582 (1276) / 60 / 89s
I	0.03125 (2520) / 24 / 1432s	0.03125 (2520) / 24 / 119s	0.003125 (252) / 240 / 119s
J	0.06250 (5040) / 24 / 725s	0.06250 (5040) / 24 / 60s	0.00625 (504) / 240 / 60s
K	0.09570 (7717) / 15 / 487s	0.09570 (7717) / 15 / 41s	0.00957 (772) / 150 / 41s
L	0.12500 (10080) / 36 / 366s	0.12500 (10080) / 36 / 30s	0.01250 (1008)/ 360 / 30s
M	0.25000 (20160) / 15 / 187s	0.25000 (20160) / 15 / 15s	0.02500 (2016) / 150 / 15s
Z	1.00000 (80640) / 2 / 100s	1.00000 (80640) / 2 / 20s	1.00000 (80640) 2 / 20s
Total Jobs / Theoretic Run Time	230 / 10773s	230 / 935s	2282 / 935s

Table 1. Synthetic workload characteristics of ESP benchmark [3, 4] and its two variations Light ESP and Parallel Light ESP x10

The typical turnaround time of normal ESP benchmark is about 3 hours, but in case of real production systems, experiments may take place only during maintenance periods and the duration of them are limited. Hence this typical problem, enabled us to define a variation of ESP called Light-ESP, where we propose a diminution of the execution time of each different class with a scope of decreasing the total execution time from 3 hours to about 15 minutes.

Table 1 provides the various jobs characteristics of ESP and Light-ESP. Each job class has the same number of jobs and the same fraction of job size as ESP but has a decreased execution time which results in a decrease of the total execution time. The execution time of each class was decreased by a factor of 0.08 except class Z which was decreased by a factor of 0.5 so that it can give a reasonable duration. The diminution of the total execution time allowed us to perform multiple repetitions of our measurements, during a relatively small duration, in order to increase our certitude about our observations.

In the case of a large cluster size like Curie (80640 cores), Light-ESP will have the 230 jobs adapted to the large system size as we can see in table 1. In order to cover different cases of workloads for the usage of such large clusters, another variation of Light-ESP seemed to be needed with a bigger number of jobs. Hence, a new variation, named Parallel Light-ESP is defined by increasing the number of all the job classes (except the Z) with a factor of 10. However, since we still want to have the same total execution time of the whole benchmark like in Light-ESP, the fraction of job size relative to system size is devided by the same factor 10. This will also allow the system to have a more adapted system utilization and probably similar fragmentation like in ESP and Light-ESP cases. Another particularity of Parallel Light-ESP is that each job class is launched by a different host allowing a simultaneous parallel submission of all type of jobs in contrast with the sequential submission of the Light-ESP. This characteristic allow us to be more closer to reality where the jobs do not arrive sequentially from one host but in parallel from multiple hosts.

The jobs in all the ESP cases are submitted to the RJMS in a pseudo-random order following a Gaussian model of submission which may be also parametrized. The tuning of the various parameters impact the inter-arrival times of the workload so various experiments allowed us to select the best fitted parameters in order to have a fully occupied system as soon as possible. In more detail, Light-ESP variation launches 50 jobs in a row (22% of total jobs) and the rest 180 jobs are launched with inter-arrival times chosen randomly between 1 and 3 seconds. In Parallel Light-ESP case each class of jobs is launched independently with similarly adapted parameters like in Light-ESP case.

Even if better alternatives than the Gaussian model of submission exist in the litterature [29, 20], we decided to keep the basis of ESP benchmark intact in this first phase of experiments. Nevertheless, our goal is to continue our studies and explore the real workload traces of "Curie" platform (when those become available). Detailed analysis of the real workload traces of the production site will allow us to perform more adapted parametrization of our proposed workloads, so that they can be as closer to reality as possible.

3.2 Experiment by Combining Real-Scale and Emulation Techniques

In order to measure the various internal mechanisms of the RJMS as one complete system we are based on real-scale experiment upon a production supercomputer and an emulation technique to validate, reproduce and extend our measurements as performed upon the real machine. Hence, at the one side we have been using subsets of "Tera-100" system during its maintenance phase to perform scalability and efficiency experiments using the described workloads; and in the same time we have been studying, analyzing and reproducing the different observations using emulation upon a much smaller fraction of physical machines.

Emulation is the experiment methodology composed by a tool or technique capable for executing the actual software of the distributed system, in its whole complexity, using only a model of the environment. The challenge here is to build a model of a system, realistic enough to minimize abstraction, but simple enough to be easily managed. Various tools currently exist for network [30] or system [31] emulation but in our case we are using simply a SLURM internal emulation technique called *multiple-slurmd*. The normal function of SLURM consists of a central controller/server daemon (slurmctld) executed upon one machine, where all the scheduling, launching and resource and management decisions take place and a compute daemon (slurmd), executed upon each computing node of the cluster. The *multiple-slurmd* technique bypasses this typical functioning model and allows the execution of multiple compute daemons slurmd upon the same physical machine (same IP address) but on different communication ports.

Hence, the number of different deployed daemon on different port will actually determine the number of nodes of the emulated cluster. Therefore the central controller will indeed have to deal with different daemons slurmd and as far as the controller's internal functions concern this emulation technique allows to experiment with them in various scales regardless the real number of physical machines.

Nevertheless, this technique has various drawbacks such us the fact that we cannot evaluate most of the functions related to the computational daemon since there may be system limitations due to the multiple number of them upon the same physical host. Furthermore, the number of internal computing resources (sockets, cores, etc) may be also emulated through the configuration files which means that task placement and binding upon the resources will not have the same performances as upon the real system and there may be limitations depending on the physical machines and file systems characteristics. In addition, we may not execute any kind of parallel application, which forces us to be limited on simple sleep jobs in order to provide the necessary time and space illusion to the controller that a real job is actually under execution. In some cases particular simple commands execution with some print into file are used in the end of the sleep period in order to be sure that the job has indeed been executed.

In order to use this type of emulation with big number of multiple-slurmd daemons upon each host some particular tuning was needed upon the physical machines especially concerning the various limits, like increasing the maximum number of user processes and the open files, using the `ulimit` command.

System scale	2012 nodes (32 cpus/node)	
Experiment method	Real NO Emulation	Emulation upon 30 nodes
Total ESP Execution time (sec)	1197	1201

System scale	4096 nodes (16 cpus/node)	
Experiment method	Emulation upon 200 nodes	Emulation upon 400 nodes
Total ESP Execution time (sec)	1283	1259

Fig. 1. Real-scale and Emulation Measurements comparisons

Finally, an important issue that had to be verified is the similarities of the experiment results between real and emulation as long as the dependence of the results into the scale of emulation (in terms of number of emulated hosts per physical machine). Theoretically speaking, concerning the RJMS scalability in terms of job launching and scheduling efficiency, these functions take place as internal algorithms of the controller daemon (slurmctld). Hence the differences between real and emulated results were expected to be trivial. Figure 1 shows the results obtained when executing Light-ESP benchmark upon real and emulated platforms of various nodes. In these experiments we obtained similar Total execution times for the same workloads which confirmed our assumptions. In addition multiple experiment repetitions made us more confident about the validity of these results.

Our experiments have been deployed upon different systems: a BULL testing cluster for internal usage and a subset of "Tera-100" system during its maintenance periods. The systems have the following hardware characteristics:

- Cuzco Cluster (BULL internal) in Clayes/France site Intel Xeon bi-CPU/quad-CORE with 20 GB memory and network structured by 1Gigabit Ethernet + Infiniband 20G.
- Subset of CEA-DAM "Tera 100" Cluster in Bruyeres-le-Chatel/France with Intel Xeon series 7500 processors (quad-CPU/octo-CORE) with 32GB of Memory and Ethernet + Infiniband networks.

4 Performance Evaluation Results

In this section we present and analyze the results of the conducted experiments in three different subsections. The results concerning the scalability of the RJMS in term of jobs number are presented first.

Then, the results and feedback of the topology aware scheduling experiments are detailed. The last subsection explains the results and the experiments realized to evaluate the scalability of SLURM in term of compute resources.

All the experiments are conducted with the consumable resources algorithm of SLURM. This algorithm enables to allocate resources at the core level and thus to share nodes among multiple jobs as long as no more than one job use a same core at the same time.

4.1 Scalability in terms of number of submitted jobs

The goal of the experiments presented in this section is to evaluate the effect of a large number of jobs on SLURM's behavior. Having a good knowledge of the system behavior in that scenario is necessary to properly configure the related limits and protect the system in its day-to-day usage.

The first set of experiments aims to stress the launch mechanism of SLURM. It enables to define the maximum submission throughput that characterizes the ability to perform a sustained rate of requests. The second set of experiments stresses the cancellation mechanism of SLURM and evaluate the level of control that can be kept on the system in borderline situations when a large amount of events has to be managed by the RJMS. The simulation of a sudden termination of the production is the exercise used for that purpose.

The experiments of the first set are realized using submission bursts of small batch jobs with a global utilization that fit in the available resources. The experiments are conducted at real-scale (no emulation) with a subset of 2020 Tera-100 nodes. The scheduler allocates resources to the jobs and starts them while receiving new requests. According to the documentation, a SLURM parameter called `defer` can be used to improve the management of this kind of workloads. The impact of this parameter is thus compared to the default behavior.

The execution time of each submitted job has to be set to a value large enough to prevent the termination of jobs before the end of the submission stage. The value used in the experiments is 500 seconds.

The results are presented in Figure 2.

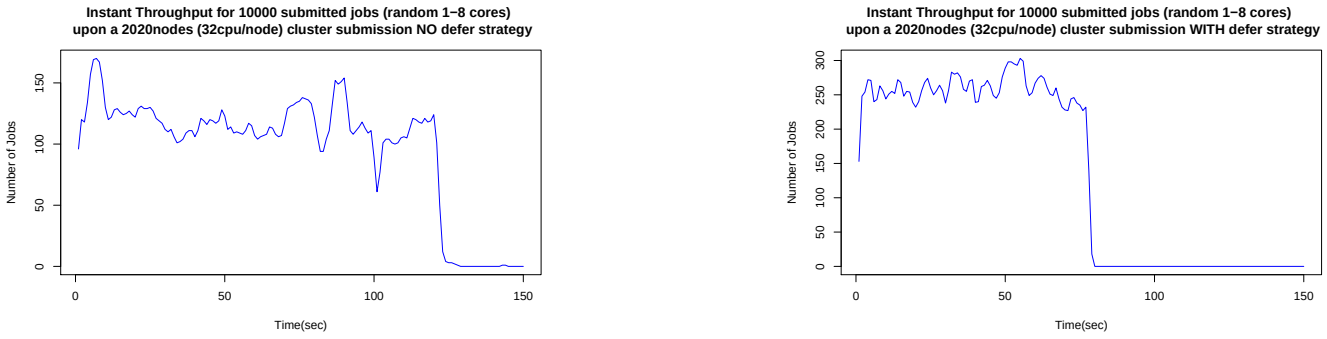

Fig. 2. Instant Throughput for 10000 simple sleep jobs (random 1 to 8 cores each) in submission upon a real cluster of 2020 nodes (32 cpus/node) with and without the usage of `defer` scheduling parameter

We see that the average submission throughput is greatly improved using the `defer` parameter. This parameter ensures that resources allocations are processed in batch by the scheduler and not sequentially at submission time. We observe an average throughput of 100 jobs per second in the standard configuration and more than 200 jobs per second when `defer` mode is activated.

We witness a gain of about 50sec for the complete submission of all the 10,000 jobs between the two cases. The main drawback of the `defer` mode is that individual jobs are slightly delayed to start. Clusters that do not have high submission bursts into their real-life workloads should not activate this parameter to provide better responsiveness.

When increasing the size of the jobs involved in these experiments, we discovered a unexpected impact on the submission throughput. As available resources were not enough to execute all the submitted jobs, the submitted jobs were put in queue in a pending state resulting in a slower pace of submission. During the slowdown, the scheduler was overloaded by the processing of the incoming submission requests. A modification of the SLURM controller was performed in order to eliminate a call to the scheduler logic while processing in parallel the incoming requests in defer mode.

The result of Figure 3 presents the instant throughput with and without this modification. In both cases, the `defer` parameter is used.

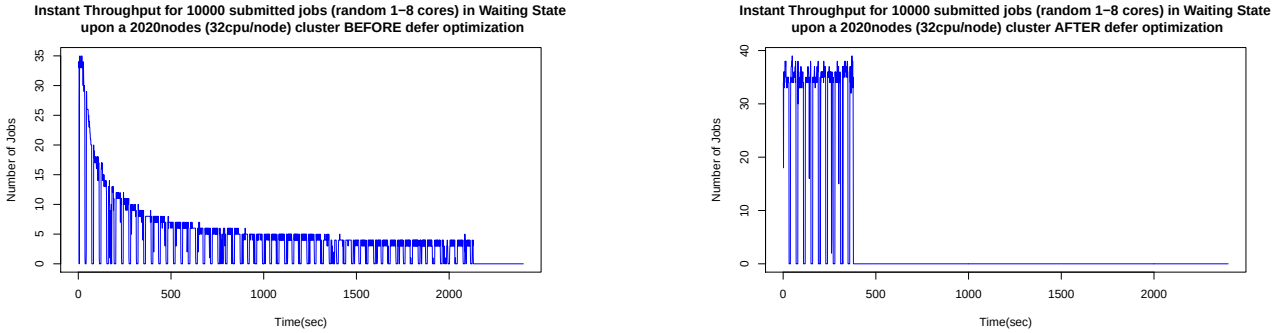

Fig. 3. Instant Throughput for 10000 simple sleep jobs (random 1 to 8 cores each) in submission in Waiting State upon a real cluster of 2020nodes (32 cpus/node) with and without an important optimization in the algorithm of `defer` scheduling parameter

We see that the modification helps to reach a constant throughput. It guarantees that only one instance of the scheduler is called regardless of the number of incoming requests. The number of jobs is thus no longer a factor of the submission rate. It can be noted that the results for waiting jobs are below the throughput of eligible jobs as evaluated above. This low limit has no link with the evaluated aspect and was corrected in a version of SLURM superior to the one used during these tests.

The experiments concerning the termination of a large workload are then performed. The initial goal is to guarantee that a large system is manageable even in the worst cases scenarios. Stopping a large number of jobs on the 2020 nodes cluster outlines a total loss of responsiveness of the system. Looking at the system during the slowdown, it is noted that the induced load is located in the consumable resources selection algorithm when dealing with the removal of previously allocated cores from the SLURM internal map of used cores. The same observation was made by a SLURM community member that proposed a patch to reduce the complexity of the algorithm used for that purpose. This patch is now part of the core version of SLURM.

The results of the termination of 10,000 jobs on the real clusters with and without this patch are presented in Figure 4.

As with the previous experiment, the patch helps to provide a constant termination throughput, no longer dependent of the number of jobs nor resources involved at the time of execution. The responsiveness of the system is now sufficient during the global termination of all the jobs.

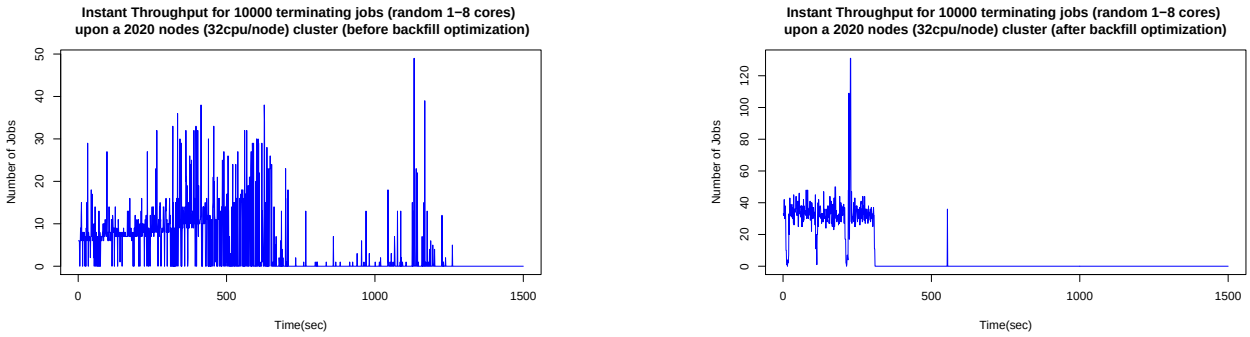

Fig. 4. Instant Throughput for 10000 simple sleep jobs (random 1 to 8 cores each) in termination upon a real cluster of 2020 nodes (32 cpus/node) before and after an important optimization upon the algorithms of backfill scheduler

Increasing number of both resources and jobs exhibits the scalability thresholds of the RJMS. The results of our experiments concerning scalability in term of jobs enable to safely consider the execution of 10,000 jobs on a production cluster using SLURM. It illustrates different issues that outline the interest of algorithms adapted to the management of high numbers of elements.

4.2 Scheduling Efficiency in terms of Network Topology aware placement

The goal of the second series of experiments is to evaluate SLURM's internal mechanism to deal with efficient placement of jobs regarding network topology characteristics, and more specifically regarding a tree network topology. SLURM provides a specific plugin to support tree topology awareness of jobs placement. The advantage of this plugin is its best-fit approach. That means that the plugin not only favors the placement of jobs upon the number of switches that are really required, it also considers a best-fit approach to pack the jobs on the minimal number of switches maximizing the amount of remaining free switches.

The experiments are focused on the evaluation of different tree topology configurations in order to select the most appropriate for "Curie" before its availability for production purposes.

The network topology used on "Curie" is a 3 levels fat-tree topology using QDR Infiniband technology. Current Infiniband technology is based on a 36 ports ASIC that made fat-tree topologies constructed around a pattern of 18 entities. The leaf level of the corresponding Clos-Network is thus made of groups of 18 nodes, the intermediate level is made of groups of 324 nodes (18*18) and the last level can aggregate up to 11664 nodes (18*18*36). The "Curie" machine is physically made of 280 Infiniband leaves switches aggregated by 18 324 ports physical switches grouping the 5040 nodes in 16 virtual intermediate switches.

The experiments are conducted using the emulation mode of SLURM on a few hundreds of Tera-100 physical nodes. These nodes were used to host the 5,040 fat-tree connected SLURM compute daemons of "Curie'. A topology description has to be provided to SLURM in order to inform the scheduler of the layout of the nodes in the topology. In emulation mode, this description is exactly the same as the description of the associated supercomputer. The real topology of the emulated cluster is different as it corresponds to the physical nodes topology. However, as the goal is to evaluate the behavior of SLURM scheduling and not the behavior of the workload execution, it is not at all an issue.

Four different topology descriptions are compared in order to evaluate the impact of the topology primitives of SLURM on the execution of a synthetic workload. The default scheduler is thus compared to the tree topology scheduler with no topology information, a medium topology information, and a fine

topology configuration. The medium description only provides information down to the intermediate level of 324 ports virtual switches. The fine description provides information down to the leaf level of ASIC of the Infiniband topology, including the intermediate level. The leaf level of ASIC is also defined here by the term "lineboards",

Global execution time as well as placement effectiveness are then compared. The placement effectiveness is calculated based on a complete description of the fat-tree topology (fine topology). This enables to measure the side effect of a non-optimal selection logic to provide optimal results.

When evaluating placement results, a job, depending on its size, can have an optimal number of lineboards, an optimal number of intermediate switches or both. For example, a job running on a single lineboard will automatically have an optimal number of lineboards and an optimal number of intermediate switches. A job running on three different lineboards of the same intermediate switch could have an optimal number of intermediate switches but a sub-optimal number of lineboards if the requested number of cores could be served by only two lineboards. A job requiring `n` lineboards could have a sub-optimal number of intermediate switches if the allocated lineboards are spread among the intermediate switches without minimizing the associated amount.

Fragmentation of the available resources over the time is one of the biggest issue of topology placement and prevents from having the theoretical placement effectiveness in practice.

The results of the topology experiments are presented in Table 2 and in Figure 5.

SLURM NB cores-TOPO Cons / Topo-ESP-Results	Theoretic- Ideal values	No Topo	Basic Topo	Medium Topo	Fine Topo
Total Execution time(sec)	925	1369	1323	1370	1315
# Optimal Jobs switches	228	53	13	180	113
# Optimal Jobs on switches + lineboards	228	21	6	120	106
# Optimal Jobs on lineboards	228	64	31	120	209

Table 2. Light-ESP benchmark results upon the emulated cluster of "Curie" with 5040 nodes (16 cpus/node), for different types of topology configuration

Ten runs of each configuration are executed and show stable results. The global execution time of the different runs are similar, having even the most detailed flavor of the tree topology strategy finishing a little bit sooner. CDF on execution times and submission times follow the same trend and show a little gain for fine topology and a little loss for the medium topology. This first result is really interesting as it validates the stability of the global throughput of the simulated clusters in regard to the different topology configurations evaluated. No trade-off between global throughput and independent jobs performances is thus required. Improvements in the jobs performances due to a better placement should directly and positively impact the system utilization.

The difference in term of placement effectiveness between the experiments without detailed topology information and the experiments with the details are huge.

In its standard behavior, the SLURM scheduler surprisingly acts better than a tree topology having all the nodes at the same level with our synthetic workload. We expected to have a same poor level of effectiveness but the best-fit logic of both schedulers treats the row of nodes differently. The original consecutive nodes allocation of the default SLURM scheduler may be at the origin of the difference.

The best placement results of the default scheduler are about 20% to 25% of right placement when only considering one of the two levels of significant topology information of the tree, and less than 10% with both simultaneously.

The medium topology configuration provides better results with about 80% of right placement for its targeted medium level and about 50% for the lineboard level and both simultaneously.

The fine topology configuration provides even better results for the lineboard level with more than 90% of right placement. The results for optimal placement at the intermediate level and for both levels simultaneously are below the previous configuration, standing between 45% and 50%.

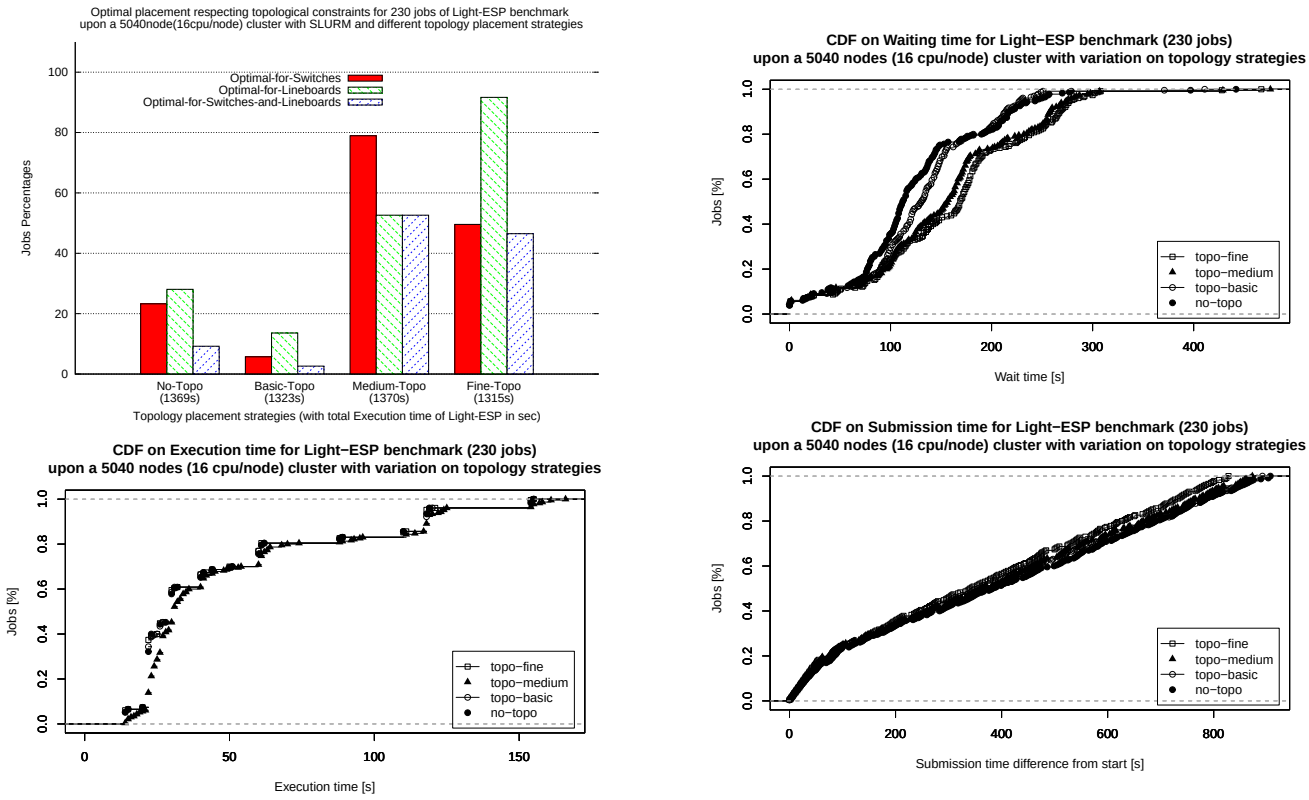

Fig. 5. System Utilization and Cumulative Distribution Functions of jobs submission and waiting times, for different cluster sizes and Light-ESPx10 with 2282 jobs

At first sight, the fine topology results are surprising as one could expect the fine strategy to be a refined version of the medium configuration only increasing the chance of having the right amount of lineboards when possible. Code inspection was necessary to understand that difference. The tree topology logic of SLURM is a two steps logic. First, it looks for the lowest level switch that satisfies the request. Then it identifies the associated leaves and do a best-fit selection among them to allocate the requested resources. Intermediate tree levels are thus only significant in the first step. The differences between the medium and the fine configuration are explained by to that logic.

When the system is fragmented and no single intermediate switch is available to run a job, the top switch is the lowest level that satisfies the request. In medium configuration, all the intermediate switches are seen as the leaves and the best-fit logic is applied against them. Chances to have the optimal number of intermediate switches are high. In fine configuration, all the lineboards are seen as the leaves and the best-fit logic is applied against them without any consideration of the way they are aggregated at intermediate levels. Chances to have the optimal number of lineboards are high but chances to have the optimal number of intermediate switches are quite low.

SLURM aims to satisfy the leaves affinity better than the intermediate levels affinity when the system is fragmented. As a result, multiple level trees are subject to larger fragmentation issues than standard 2-levels tree. Indeed, the leaves being equally treated when fragmentation occurs, jobs are spread among the intermediate level switches without any packing strategy to increase the chance of recreating idle intermediate switches. Considering that fact, the choice between using a medium versus a fine topology configuration is driven by both the hardware topology and the workload characteristics.

Pruned fat-tree topologies, like the one used in Tera-100, have smaller bandwidths between switches located at the intermediate level. Using a fine configuration with these topologies could result in the incapacity to provide enough bandwidth to jobs requiring more than one switch because of too much fragmentation. A medium configuration, only focusing on detailed information about the pruned level is certainly the best approach.

For systems where the workload characteristics let foresee idle periods or large job executions, then the fine configuration is interesting. Indeed, it will favor the optimal number of lineboards and even switches when the system is not highly fragmented. If the workload characteristics let foresee jobs having the size of the intermediate switches or if a high 24/24 and 7/7 usage would inevitably conduct to a large fragmentation, using the medium configuration appears to be a better approach.

The results of the topology experiments are really interesting to help configuring the most adapted topology configuration for tree based network topology. A same approach, using a synthetic workload tuned to reflect a site typical workload should be used in addition to confirm the choice before production usage. It could also be used on a regular basis to adapt the configuration to the evolution of jobs characteristics.

SLURM behavior with multiple-level trees needs to be studied deeper in order to determine if a balanced solution between medium and fine configuration could bring most of the benefits of each method and enable to have better effectiveness with a fine description. A best-fit selection of intermediate switches in the first step of SLURM tree topology algorithm will have to be evaluated for that purpose.

4.3 Scalability in terms of number of resources

The light-ESP benchmark [32], described in section 3, is used in these experiments. This synthetic workload of 230 jobs with different sizes and target run times is run on different emulated clusters from 1,024 nodes to 16,384 nodes. In this benchmark, the size of the jobs automatically grows with the size of the targeted cluster. As a result, a job size always represent the same share of the targeted system.

To evaluate the job size effect on the proper execution of the workload, a modified version of the Light-ESP benchmark having 10 times the number of 10x smaller jobs is used. This benchmark results in the same amount of required resources and total execution time as with the previous workload.

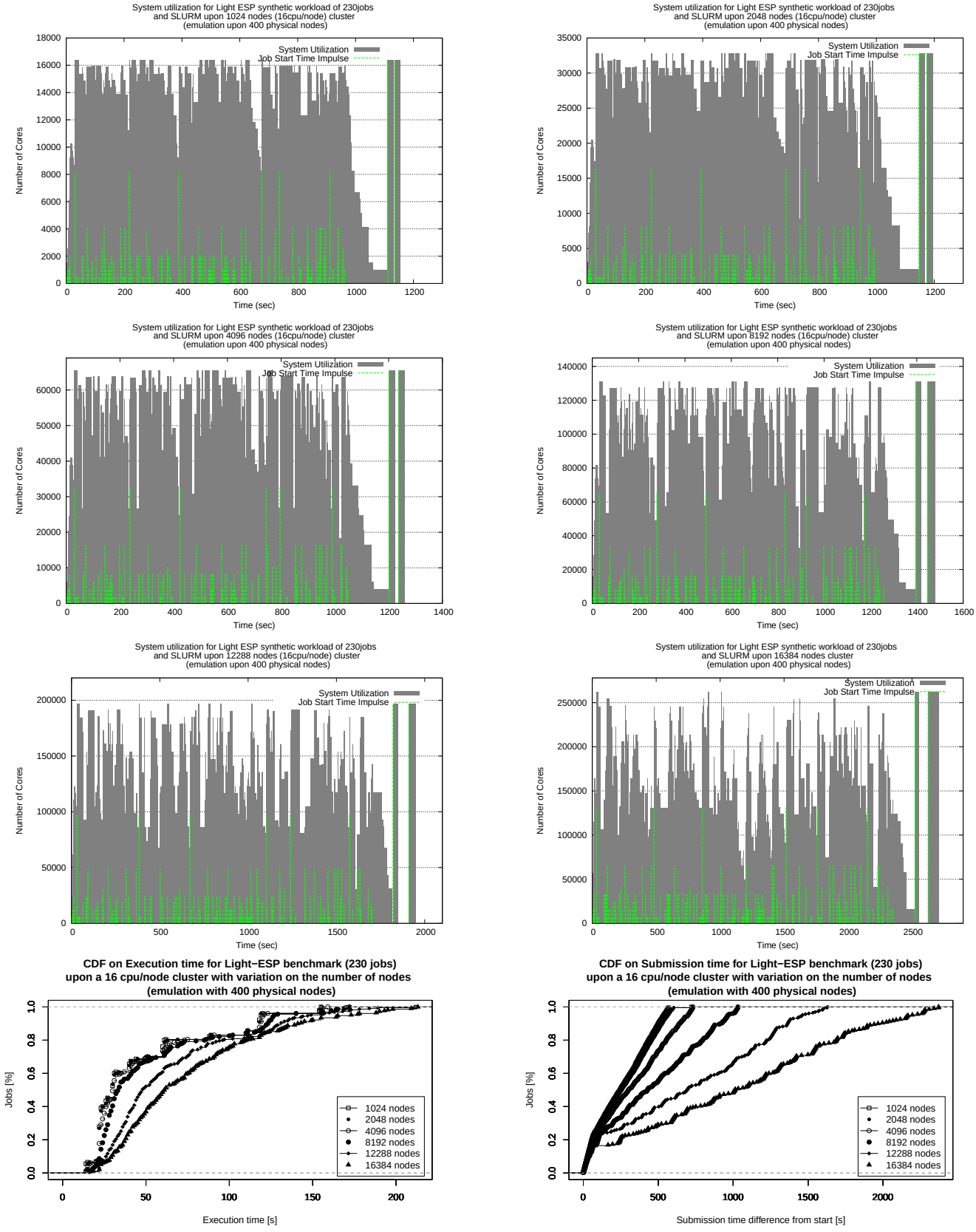

Fig. 6. System Utilization and Cumulative Distribution Functions of jobs submission and execution times, for different cluster sizes and Light-ESP with 230 jobs

In all cases, a single encapsulated SLURM client call is used in every job in order to evaluate the responsiveness of the system. The more the central controller is loaded, the more the execution time is increased by the delay of the associated reply reception. The results gathered during the executions of the Light-ESP for different cluster sizes are detailed in Figure 6.

We see an increase in jobs execution times as well as a stretching of CDF on submission time for clusters of 8k nodes and more. The results suggest a contention proportional to the number of compute nodes in the central controller process. Looking closer at the system during a slowdown, the central SLURM controller is overloaded by a large amount of messages to process. The messages are mostly internal messages that are transmitted by the compute nodes at the end of each job.

Current design in SLURM protocol (at least up to version 2.3.x) introduces the concept of epilogue completion message. Epilogue completion messages are used on each node involved in a job execution to notify that the previously allocated resources are no longer used by a job and can be safely added to the pool of available resources.

This type of message is used in a direct compute node to controller node communication and result in the execution of the SLURM scheduler in order to evaluate if the newly returned resources can be used to satisfy pending jobs. In order to avoid denial of service at the end of large jobs, SLURM uses a dedicated strategy to smooth the messages throughput over a certain amount of time. The amount of time is capped by the number of nodes involved in the job multiplied by a configuration variable defining the estimated process time of a single message by the controller. This strategy induces latencies at the end of the jobs before the global availability of all the associated resources for new executions.

In SLURM vocabulary, these latencies correspond to the time the previously allocated nodes remain in the `completing` state. The decreasing system utilization densities of the workloads when cluster size increases, as displayed in Figure 6, outline these latencies.

Each Light-ESP workload is finished when two jobs using all the resources are processed. As a result, the end of each workload enables to have a direct picture of the effect of the completing latency. While the number of nodes increases on the emulated clusters, the idle time between the last two runs increases proportionally. `EpilogMsgTime`, the parameter used to define the amount of time of a single epilogue complete message processing, has a default value of 2ms. This result in a completing window of at least 32s when a 16,384 nodes job is finishing. Looking at the results of our experiments, it sounds that the delay between the last two jobs is even greater than the expected completing time. In order to understand that behavior, a new set of experiments are conducted. The associated results are shown in Figure 7.

The idea was to first reduce the EpilogMsgTime in order to see if the completing time would be proportionally reduced and in a second phase to reduce the epilogue message processing complexity to see if the time to treat each message would be reduced too. For the second target, a patch was added in order to remove the scheduling attempt when processing an epilogue complete message on the controller. The epilogue message time was kept smaller for this run too in order to have a better idea of the additional effect of the two propositions. As shown by Figure 7, the two strategies enable to reduce the global computation time consecutively and provide reduced execution time in comparison with the standard run. However, the results are still no longer as effective as for small clusters. The CDF on submission times on the same figure still shows the stretching of the curves that reflects delay of submissions of the ESP launcher. The CDF on execution time shows an increase of execution times that suggests an unresponsive controller during the jobs execution for large clusters. The epilogue completion mechanism in SLURM, and more specifically its processing time and its peer-to-peer communication design appears to be the main bottlenecks at scale.

Further investigations must be done in order to evaluate the feasibility of a modification of this protocol in order to achieve better performances. Reversed-Tree communications, like the one used for the aggregation of jobs statistics of each node right before the epilogue stage could be a good candidate. This would result in spreading the overhead of messages aggregation to the compute nodes and let the

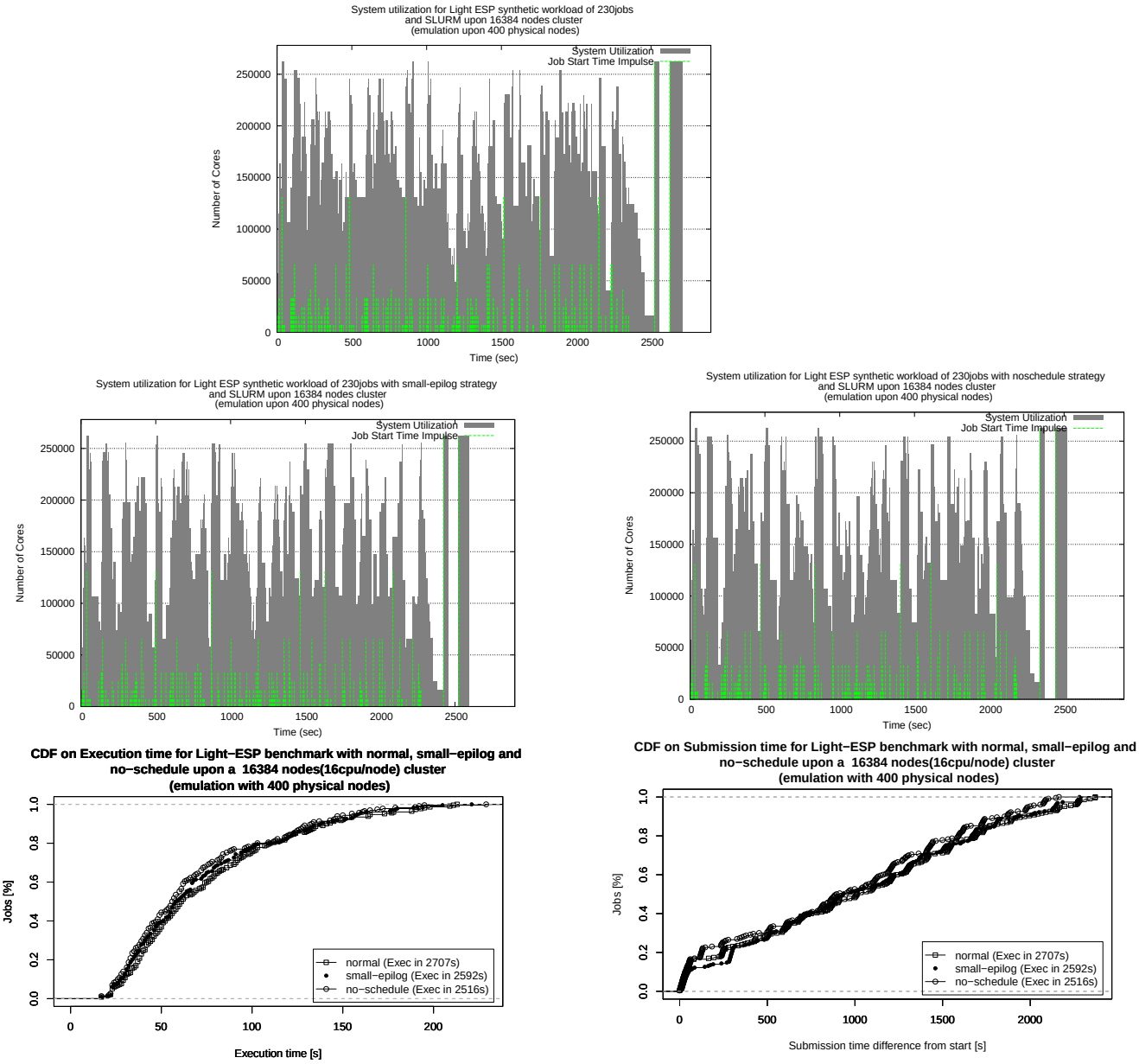

Fig. 7. Cumulative Distribution Functions of jobs submission and execution times, for different termination strategies upon a 16384 nodes(16 cpus/node) cluster and Light-ESP with 230 jobs

central controller only manage one single message informing of the completion of the epilogue at the end of each job. Side effects, like compute nodes taking too much time in processing the epilogue stage will have to be taken into account. This issue may be one of the origins of the peer-to-peer design choice that is made for current version of SLURM. However, the previous results outline the fact that large clusters require a new balance between epilogue completion aggregation and quick return to service of previously allocated nodes.

Large jobs induce long completing periods and a high load of the controller because of large number of epilogue completion message to process. It is interesting to evaluate the behavior of SLURM with the same different cluster sizes but with a larger workload composed of smaller jobs to determine if this behavior is still observed. The results of the experiments are presented in Figure 8.

For small clusters, up to about 4,096 nodes, the global throughput of such a workload is really great and even close to the theoretical value of 935 seconds. An interesting result is that the responsiveness of the controller is also greater. Indeed, there is no longer a large variation on the CDF on execution times for the different cluster sizes. This has to be related to the smoothing of the job endings induced by the larger workload of smaller jobs. However, large clusters, having the ability to run a larger numbers of jobs simultaneously, are subject to the same kind of stretching of system utilization than with the original Light-ESP workload. The effect is even worst as the system utilization is not only stretched but collapses over the time.

Based on the feedback of the previous studies on large number of job submissions and the effect of removing the scheduler call in the epilogue complete processing of the controller, a potential candidate to explain the effect would be the impact of SLURM internal scheduler when called at submission and completing time. Indeed, the CDF on submission time shows that the submissions on the larger clusters are delayed during the execution of the workloads on the large supercomputers. As the responsiveness stays similar across the different sizes, the collapses should be due to the scheduling complexity at submission time that decrease the submission rate.

Further experiments are required to identify the reasons of this issue. The usage of the `defer` parameter, as well as the patches made to remove the various useless scheduler call when this mode is activated, will have to be tested to validate this assumption.

To sum up, the results on the scalability in terms of number of resources show that good efficiency and responsiveness can be obtained for clusters up to 4,096 nodes. The scalability threshold is placed between 4,096 and 8,192 nodes in the evaluated configuration with the consumable resources selection algorithm of SLURM. A first way of enhancement has been identified and should help to provide better result in term of scalability for small workload (in number of jobs). A second issue has been identified and could prevent from extending the positive effect of the first one to larger workload (in number of jobs).

Improvements and new experiments are required to prepare SLURM to manage larger clusters. Current production clusters should not suffer of the outlined issues but must be monitored.

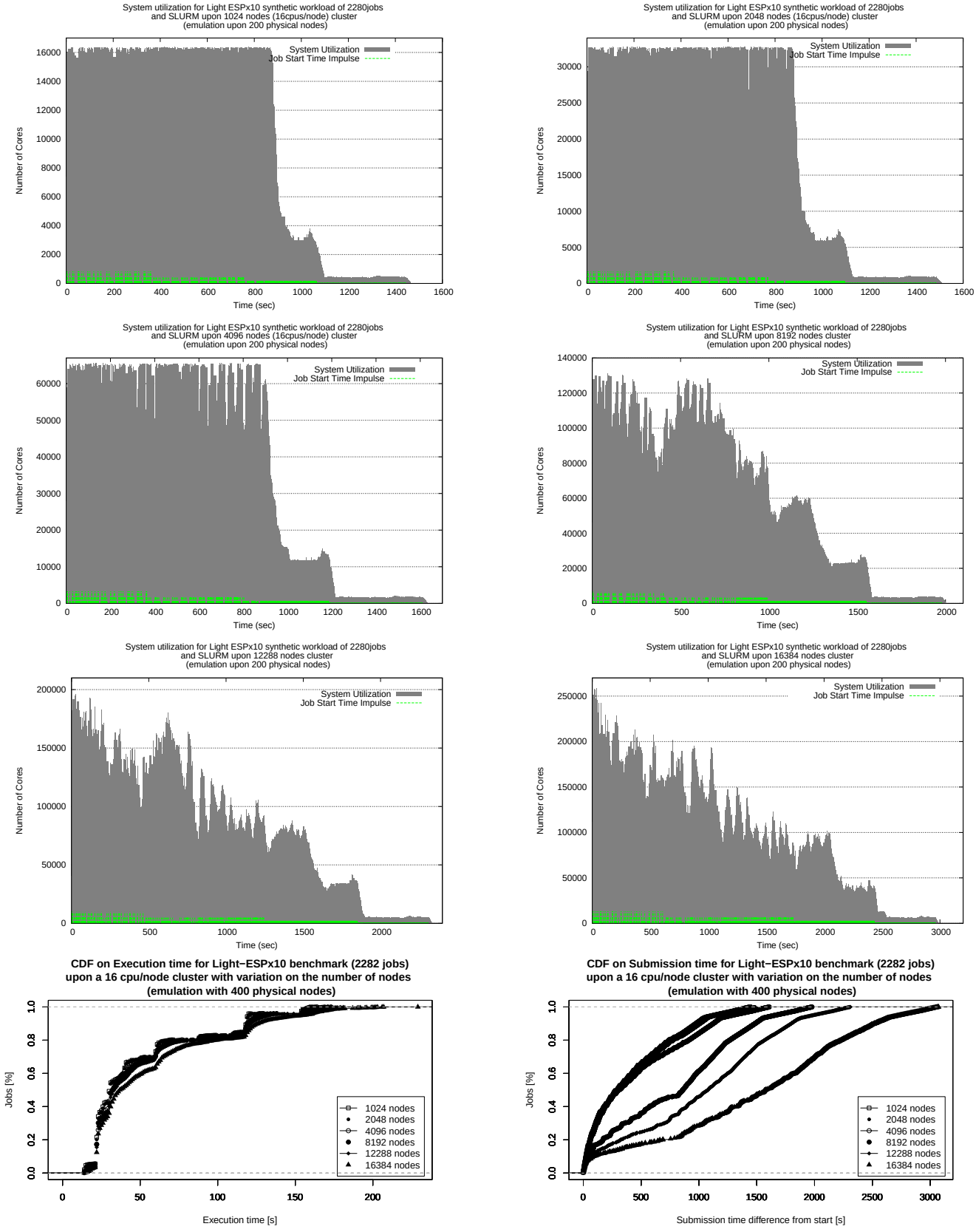

Fig. 8. System Utilization and Cumulative Distribution Functions of jobs submission and execution times, for different cluster sizes and Light-ESPx10 with 2282 jobs

5 Conclusions and future works

In this paper we have presented an evaluation methodology for the Resource and Job Management System SLURM based upon combined real-scale and emulation experiments using synthetic workloads.

We have validated our methodology by observing similar results for identical workloads on both the real-scale and emulated systems. Using emulated clusters is a real alternative to cost-effective real-scale experiments that are not only hard to be approved but also suffers from the inherent hardware faults that make evaluating a system in its complete availability a pretty hard task. The validation of this methodology enabled us to conduct experiments on emulated clusters with a strong confidence on their correctness in real situation.

The choice of using synthetic workloads was initially performed to ease the modification of their characteristics and address a broader spectrum of scenarios. Furthermore, the PRACE targeted machine, "Curie", not being in production at the petaflop scale at the time of the study, no real workload traces were available as an input for a similar approach.

The conducted experiments helped to guarantee the capabilities and the efficiency of the open source RJMS compared to the requirements of the targeted petaflop machines, "Tera-100" and "Curie". In a second time, experiments at larger scales were realized to not only have a precise idea of the behavior of the product on current machines but also evaluate the capability of SLURM to manage clusters up to 16,384 nodes.

The conclusions raised during the evaluations are significant for the day-to-day usage of the installed machines and let us have a better view of the work that will be required in the following months and years to prepare a valid RJMS candidate for the next generation of supercomputers. Our plans are now to extend our methodology with real workload traces replay to confirm our first assumptions concerning the configuration parameters to use on our system. In addition, we will work on the correction of the identified topology awareness and scalability limitations in order to enhance SLURM for the proper management of clusters up to 16,384 nodes.

The emulation experience acquired during this study let us consider that an emulated cluster of up to 65,536 nodes, the theoretical limit of SLURM, could be correctly emulated on the currently available petaflop machines. We will certainly face new issues and thresholds associated to such a new scale of experiments.

References

1. Top500 supercomputer sites. http://www.top500.org/.
2. Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple Linux utility for resource management. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer Verlag, 2003. Lect. Notes Comput. Sci. vol. 2862.
3. Adrian T. Wong, Leonid Oliker, William T. C. Kramer, Teresa L. Kaltz, and David H. Bailey. System utilization benchmark on the cray T3E and IBM SP. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing, IPDPS 2000 Workshop, (6th JSSPP 2000)*, volume 1911 of *Lecture Notes in Computer Science (LNCS)*, pages 56–67, Cancun, Mexico, May 2000. Springer-Verlag (New York).
4. William TC Kramer. *PERCU: A Holistic Method for Evaluating High Performance Computing Systems*. PhD thesis, EECS Department, University of California, Berkeley, Nov 2008.
5. Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: A load sharing facility for large, heterogeneous distributed computer systems. Technical report, 1993.
6. Ibm loadleveler. http://www.redbooks.ibm.com/redbooks/pdfs/sg246038.pdf.
7. Robert L. Henderson. Job scheduling under the portable batch system. In *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 279–294, London, UK, 1995. Springer-Verlag.
8. Moab workload manager. http://www.adaptivecomputing.com/resources/docs/mwm/7-0/help.htm.

9. Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.
10. Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard. A batch scheduler with high level components. In *5th Int. Symposium on Cluster Computing and the Grid*, pages 776–783, Cardiff, UK, 2005. IEEE.
11. Grid engine. http://gridscheduler.sourceforge.net/howto/howto.html.
12. Torque resource manager. http://www.adaptivecomputing.com/resources/docs/torque/4-0/help.htm.
13. Maui scheduler. http://www.adaptivecomputing.com/resources/docs/maui/index.php.
14. Joseph A. Kaplan and Michael L. Nelson. A comparison of queueing, cluster and distributed computing systems. NASA TM-109025 (Revision 1), NASA Langley Research Center, Hampton, VA 23681-0001, june 1994.
15. Mark A. Baker, Geoffrey C. Fox, and Hon W. Yau. Cluster computing review, 1995.
16. Tarek A. El-Ghazawi, Kris Gaj, Nikitas A. Alexandridis, Frederic Vroman, Nguyen Nguyen, Jacek R. Radzikowski, Preeyapong Samipagdi, and Suboh A. Suboh. A performance study of job management systems. *Concurrency - Practice and Experience*, 16(13):1229–1246, 2004.
17. Walfredo Cirne and Francine Berman. A comprehensive model of the supercomputer workload. In *4th Workshop on Workload Characterization*, pages 140–148, December 2001.
18. Eitan Frachtenberg and Uwe Schwiegelshohn. New challenges of parallel job scheduling. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–23. Springer Verlag, 2007. Lect. Notes Comput. Sci. vol. 4942.
19. Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing, 13th IPPS/10th SPDP'99 Workshop (5th JSSPP'99)*, volume 1659 of *Lecture Notes in Computer Science (LNCS)*, pages 67–90, San Juan, Puerto Rico, USA, April 1999. Springer-Verlag (Berlin).
20. Eitan Frachtenberg and Dror G. Feitelson. Pitfalls in parallel job scheduling evaluation. In Dror G. Feitelson, Eitan Frachtenberg, Larry Rudolph, and Uwe Schwiegelshohn, editors, *JSSPP*, volume 3834 of *Lecture Notes in Computer Science*, pages 257–282. Springer, 2005.
21. Dror G. Feitelson. Metric and workload effects on computer systems evaluation. *IEEE Computer*, 36(9):18–25, 2003.
22. Abhinav Bhatele, Eric J. Bohm, and Laxmikant V. Kalé. Topology aware task mapping techniques: an api and case study. In *PPOPP*, pages 301–302, 2009.
23. C. E. Leiserson. Fat-trees: Universl networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, c-34(10), October 1985.
24. Javier Navaridas, José Miguel-Alonso, Francisco Javier Ridruejo, and Wolfgang Denzel. Reducing complexity in tree-like computer interconnection networks. *Parallel Computing*, 36(2-3):71–85, 2010.
25. Bay and Bilardi. Deterministic on-line routing on area-universal networks. *JACM: Journal of the ACM*, 42, 1995.
26. Eitan Frachtenberg, Fabrizio Petrini, Juan Fernández, and Scott Pakin. Storm: Scalable resource management for large-scale parallel computers. *IEEE Trans. Computers*, 55(12):1572–1587, 2006.
27. Juan Fernández, Eitan Frachtenberg, Fabrizio Petrini, and José Carlos Sancho. An abstract interface for system software on large-scale clusters. *Comput. J.*, 49(4):454–469, 2006.
28. Ioan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster, and Mike Wilde. Falkon: a fast and light-weight task execution framework. In *IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC07*, 2007.
29. Uri Lublin and Dror G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63:2003, 2001.
30. Kashi Venkatesh Vishwanath, Amin Vahdat, Ken Yocum, and Diwaker Gupta. Modelnet: Towards a datacenter emulation environment. In Henning Schulzrinne, Karl Aberer, and Anwitaman Datta, editors, *Peer-to-Peer Computing*, pages 81–82. IEEE, 2009.
31. Louis-Claude Canon and Emmanuel Jeannot. Wrekavoc: a tool for emulating heterogeneity. In *IPDPS*. IEEE, 2006.
32. Adrian T. Wong, Leonid Oliker, William T. C. Kramer, Teresa L. Kaltz, and David H. Bailey. ESP: A system utilization benchmark. In ACM, editor, *SC2000: High Performance Networking and Computing. Dallas Convention Center, Dallas, TX, USA, November 4–10, 2000*, pages 52–52, pub-ACM:adr and pub-IEEE:adr, 2000. ACM Press and IEEE Computer Society Press.