Computer Science • 13 (2) 2012 http://dx.doi.org/10.7494/csci.2012.13.2.5

Václav Chlumský
Dalibor Klusáček
Miroslav Ruda

THE EXTENSION OF TORQUE SCHEDULER ALLOWING THE USE OF PLANNING AND OPTIMIZATION IN GRIDS

Abstract *In this work we present a major extension of the open source TORQUE Resource Manager system. We have replaced a naive scheduler provided in the TORQUE distribution with complex scheduling system that allows to plan job execution ahead and predict the behavior of the system. It is based on the application of job schedule, which represents the jobs' execution plan. Such a functionality is very useful as the plan can be used by the users to see when and where their jobs will be executed. Moreover, created plans can be easily evaluated in order to identify possible inefficiencies. Then, repair actions can be taken immediately and the inefficiencies can be fixed, producing better schedules with respect to considered criteria.*

Keywords grid, scheduling, TORQUE, schedule, planning, predictability, evaluation

1. Introduction

In this work we present an extension of the open source TORQUE Resource Manager (RM) [4, 26], which is used in the Czech National Grid Infrastructure *MetaCentrum* [27]. Nowadays, all major production resource management systems such as PBS Pro [14], LSF [36], Sun Grid Engine (SGE) [11], TORQUE, etc. use the classical queueing approach when scheduling jobs on the resources. On the other hand, in the past decade many works have shown that the use of a planning represents several advantages [34, 21, 28]. Unlike the queueing approach where scheduling decisions are taken in an ad hoc fashion often disregarding previous and future scheduling decisions, planning-based approach allows to make plans concerning job execution. The use of such a plan (job schedule) brings several benefits. It allows us to make a prediction of job execution providing the user information concerning the expected start time of their jobs thus improving predictability [28]. Moreover, the prepared plan can be evaluated with respect to selected optimization criteria, using proper objective functions. Then, it can be optimized with some advanced scheduling technique such as Local Search in order to improve the schedule's quality. However, as far as we know there is no working implementation of such a plan-based scheduler in a production open source resource management system. Therefore, we have decided to develop a plan based scheduler in the TORQUE Resource Manager that would allow such a functionality.

So far we have developed a working implementation of job schedule that can be used to plan job execution onto one or more computing sites such as a computer cluster. The schedule is created subject to dynamically arriving jobs. Then, it is used to schedule jobs on the available computing resources. The schedule is continuously maintained in order to remain consistent with the changing situation in the system. Thus, all important events such as job arrivals and (early) job completions or machine failures and restarts are reflected in order to keep the schedule up-to-date with respect to the changing situation. Moreover, the users can now *query the scheduler* to get information from the job schedule. It means that they can ask the scheduler *when and where* their jobs will be executed and the scheduler provides them such a prediction according to the current job schedule. Also, several evaluation criteria has been implemented allowing for the use of schedule optimization techniques.

The structure of this paper is as follows. In the next section we introduce some basic terminology and we discuss the related work. In Section 3 we describe the extension of the TORQUE Resource Manager, describing the details of schedule implementation (Section 3.3). Also, the crucial methods needed to construct and maintain the schedule are presented (Section 3.4). Next, an experimental evaluation of the schedule-based solution is presented, discussing the performance of the applied solution. Finally, we conclude the paper and we discuss future work.

2. Related work

2.1. Queue-based systems

All major production systems such as PBS Pro [14], LSF [36], Sun Grid Engine (SGE) [11], Condor [32], Maui and Moab [3] as well as higher level metascheduling systems such as GridWay [13], gLite WMS [7], QCG-Broker [23], etc., are so called queueing systems. It means that these systems typically follow the queue-based scheduling approach, using one or more incoming *queues* where jobs are stored until they are scheduled for execution. A scheme of a basic (local) queueing system is shown in Fig. 1 (left).

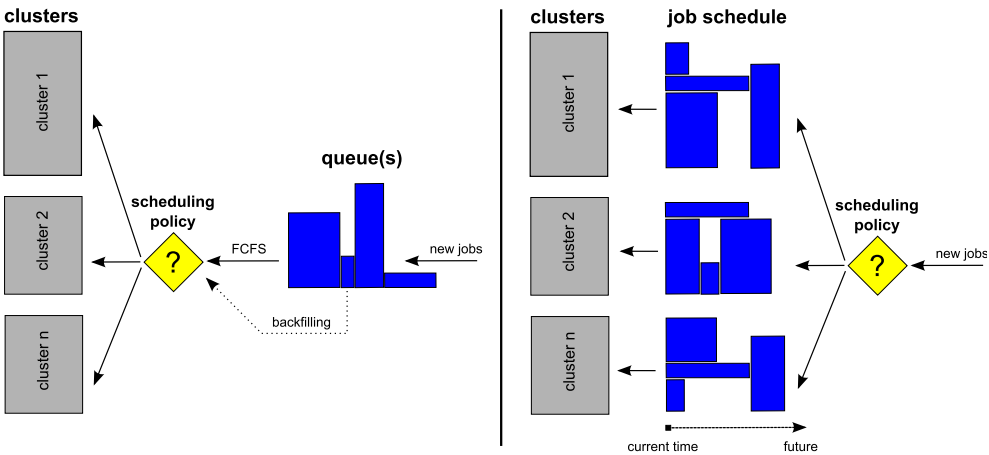

Figure 1. The scheme of queue-based (left) and schedule-based (right) systems

The scheduling process consists of several steps that define the behavior of the whole scheduler. Upon each job arrival a queue and a position among other jobs is established. If there are more queues in the system, the proper one is selected[1]. The order of the jobs in a given queue is either based on a *First Come First Served (FCFS)* principle or some priority-based ordering can be applied [16]. Once the ordering is established, the scheduling policy either attempts to run one job from each queue or all jobs from the currently selected queue are checked before the next queue is processed. Also, the combination of these approaches can be used [14].

The job is selected for execution if all the resources it requires are currently available on some cluster. The decision is made by a *scheduling policy* which is typically some form of heuristic algorithm which selects jobs from the queue for execution. Within a given queue, various scheduling policies can be used to decide which job will be scheduled. Frequently, either a simple *First Come First Served (FCFS)*, some priority based policy such as *Shortest Job First* [14], *Earliest Deadline First* [6], or some

[1]Typically, the target queue is directly specified upon each job submission.

form of *backfilling* [29, 28] are typically used for this purpose. With the exception of backfilling, where reservations can be used, the final scheduling decision is often made at the very last moment — once some machine(s) become available. Therefore, the scheduling is done in an ad-hoc fashion with limited consideration of previous or future decisions.

2.2. Schedule-based systems

Schedule-based systems represent a significantly different approach with respect to the queue-based solutions. Schedule-based systems use job *schedule* as a more complex data structure that *maps jobs onto available machines in time* [2, 12]. This schedule represents de facto a plan of future job execution. In the same fashion as, e.g., backfilling algorithms require information about expected job runtime, also schedule-based solutions need such information to construct the schedule. A scheme of the general schedule-based system is shown in Fig. 1 (right).

Clearly, there are no incoming queues that would store the jobs [12]. Instead of that a two dimensional data structure is being built that assigns to each job its own space and time slot on a given machine. The x-axis represents system time and the y-axis represents a particular machine on a particular cluster. There are other aspects that differentiate the design of schedule-based systems from queue-based systems. The major difference is that a nontrivial scheduling decision must be taken every time some new job arrives. Such an immediate decision is necessary to find a suitable place for the job in the schedule. As discussed in Section 2.1, queue-based methods usually perform such decision when the job is selected for execution, i.e., at the "last possible moment" (when machine(s) become available). An exception represents those queue-based methods that use reservations. When machine(s) become available, scheduling decisions are trivial for schedule-based methods. At that point in time, a scheduler simply sends on such machine(s) those jobs that are stored in the schedule on corresponding coordinates.

The use of a schedule offers three major advantages. First, the schedule allows us to *make predictions*, i.e., to guarantee for each job its start time, expected completion time, and the target machine(s). Such information can be very valuable to users since they can use it to better organize and plan their work [10]. Second, since the schedule holds detailed information about the expected execution time of each job, it is possible to *evaluate the quality of the solution* which is represented by such a schedule. None of these functionalities are typically available when classical queueing systems are used as this information is not known in advance. Several criteria can be used [22] for evaluation, allowing us to identify possible problems concerning job performance, the efficiency of resource utilization or, e.g., fairness issues. Next, an *optimization procedure* can be launched that tries to improve the quality of the schedule with respect to the applied optimization criteria [21, 22].

As far as we know, no major production system works by default in a schedule-based mode. An experimental *Computing Center Software (CCS)* scheduling sys-

tem [15] uses a schedule-based approach. However, the schedule is only used for prediction and no evaluation or optimization has been implemented [12]. In [31], another experimental schedule-based system designed for scheduling sequential and parallel jobs as well as workflows has been presented. Sadly, these systems are either no longer operational or they represent proprietary solutions which are not freely available. On the other hand, many current systems [24] support so called *advanced reservation (AR)* [30, 24], where jobs can obtain reservations based on the users' requests. However, the use of ARs represent several problems. Usually, only a fraction of jobs/users is allowed to request ARs. Then, problems occur as guaranteeing ARs may cause degradation of the overall performance [30]. Next, as the runtime estimates of non-AR jobs are usually significantly longer than their actual runtime, idle periods often appear before reservations. As ARs are typically fixed, it may not be possible to fill appearing gaps with non-AR jobs (too long estimates) and the performance degrades again [24]. As a result, the use of advanced reservations is often restricted by the system administrators.

From this point of view we distinguish between an AR-enabled system and a truly schedule-based system. The first difference is that in schedule-based system *every job* gets a "reservation". Second, this "reservation" is not strict and it can be shifted to an earlier as well as to a later time, if desirable. It allows us to perform more aggressive improvements, e.g., via metaheuristics. These changes are controlled by the evaluation procedure to guarantee good quality of the resulting solution. There are several works that propose the applications of schedule-based approaches in the literature [1, 2, 25, 34]. A nice survey of recent applications can be found in [35].

3. Extension of the TORQUE Resource Manager

The major result of this work is the development of a new scheduler in a production open source resource management system that uses job schedule to plan job execution. In the following text we describe the implementation details of our solution.

3.1. TORQUE Resource Manager

TORQUE Resource Manager is an advanced open-source product based on the original PBS project which provides control over batch jobs and distributed computing resources. TORQUE consists of three main entities — the server (`pbs_server`), the scheduler (`pbs_sched`), and the node daemons (`pbs_mom`). The scheduler makes the scheduling decisions and interacts with the `pbs_server` in order to allocate jobs onto available nodes. In our work, we have replaced the original simple queue-based FCFS scheduler [4] available in the TORQUE's scheduler entity (`pbs_sched`). The remaining entities such as Server (`pbs_server`) or node daemons (`pbs_mom`) are mostly unchanged. The solution builds and manages the job schedule according to the dynamically arriving events from the `pbs_server`. One schedule is created for each virtual or physical site (e.g., computer cluster). The design of the schedule data structure was the key problem here and we describe it in the following section.

3.2. Problem description

The design of the schedule data structure is the key factor. Let us first formally describe the requirements concerning the schedule data representation. We consider n jobs and a Grid system that consists of r nodes that all together have m CPUs. Each node k has a fixed amount of available RAM memory RAM_k. Each job j is characterized by its processing time p_j, estimated processing time ep_j, by the number of requested CPUs $usage_j$ and by the amount of requested RAM mem_j. In reality, the Grid typically consists of one or more sites such as computer clusters. For each cluster, we would create a separate instance of schedule. For simplicity let us assume that the Grid consists of a single cluster so there is only one schedule instance. Intuitively, the schedule defines *when and where* jobs will be executed, introducing a two dimensional rectangular representation such that for each job the set of assigned CPUs and the time interval is specified. Here the x-axis represents the time and the y-axis represents the CPUs of the system. An example of such a schedule is shown in Fig. 2 (left).

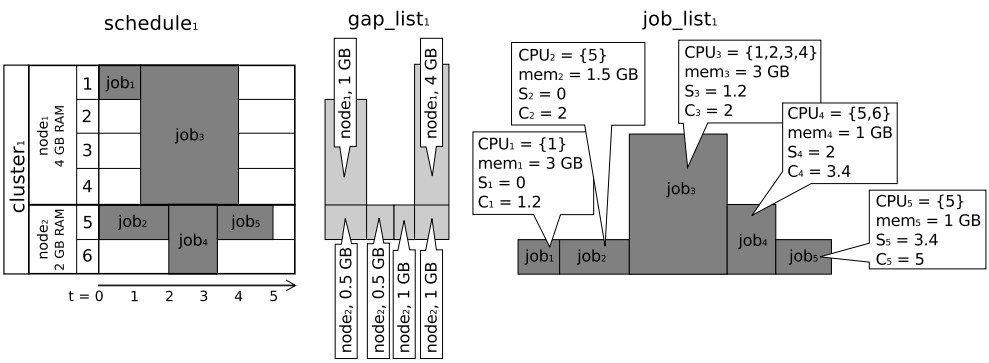

Figure 2. Original schedule (left) as represented by *gap_list* (middle) and *job_list* (right)

Formally, the schedule assigns each job j a set of CPUs ($\mathcal{CPU}_j$) it will use, where the size of $\mathcal{CPU}_j$ is equal to $usage_j$. The schedule also specifies the time interval during which the job j will be executed. This interval is denoted by the expected start time S_j and completion time C_j. We do not consider job preemption, therefore $C_j - S_j = p_j$ holds. Clearly, if the executions of two different jobs j_1 and j_2 overlap in time then $\mathcal{CPU}_{j_1} \cap \mathcal{CPU}_{j_2} = \emptyset$. Furthermore, let us assume that at some point in time jobs $j_1, j_2, ..., j_p$ are executed on node k at the same moment. Then, $\sum_{i=1}^{p} mem_{j_i} \leq RAM_k$. This formula guarantees that the amount of available RAM on given node k is greater than or equal to the amount of requested RAM of jobs running on that node k. We now proceed to the description of the applied data representation.

3.3. Data representation of job schedule

In [28], so called *profile of free CPUs* represented as a linked list is used to determine whether jobs from a queue can be backfilled without delaying the execution of some older waiting jobs. A similar structure is often used for cumulative scheduling [5]. In our recent work [20] we have proposed a representation, where the schedule is represented as a linear list of jobs (*job_list*) and a list of so called *gaps* (*gap_list*) that represents the free CPUs. The *gap_list* is an analogy of the profile used in [28]. Gaps and jobs in these lists are ordered according to their expected start times. Each job in this list stores its S_j, C_j, $\mathcal{CPU}_j$. Therefore, complete information as required by the definition in Section 3.2 is stored in a single cell of the list. All these parameters are computed when the job is added into the schedule. Of course, because the schedule can change dynamically in time, the parameters can be lately updated according to the new situation.

Gaps represent unused periods of CPU time and the amount of free RAM across nodes within a cluster. Gaps appear every time the number of available CPUs in the existing schedule is greater than the number of CPUs requested by the job(s) in the given time period. Similarly, a gap can appear when the amount of requested RAM is higher than is currently available in a given time period[2]. Similar to jobs, gaps hold information about their start time, duration, usage, and free RAM. Here, the usage expresses the number of available (idle) CPUs in this gap. It is stored as an array of such available CPUs with a pointer to appropriate node and the node's free RAM. Moreover, each gap has a pointer to the nearest following job. If there is no following job (gap at the end of the *schedule*) this pointer is set to *null*. For given time t, there is always at most one gap in the *gap_list*. Two successive gaps in the *gap_list* have either different usage, contain different nodes, contain different amounts of free RAM for the given node or the completion time of the earlier gap is smaller than the start time of the later gap. Otherwise, such gaps are merged into a single, longer gap. Fig. 2 (right) shows an example of the applied data structure that represents the schedule. As discussed in Section 3.2, only one cluster is considered for simplicity, i.e., only one schedule is shown in the figure.

3.4. Schedule-related auxiliary procedures

The proposed schedule structure is used by several procedures that are used either to retrieve information from the schedule or to keep the schedule up-to-date subject to the dynamically changing state of the system.

A backfill-like procedure is used to *build the schedule*, i.e., to add newly arriving job into the currently known schedule. It finds the earliest suitable gap for the new jobs [21]. This is a common scenario used in several popular algorithms such as in EASY or Conservative backfilling [28]. In this case, the applied data representation

[2]In such situation two or more jobs cannot be planned to the same time period and some of them must be delayed to a period when enough RAM is available. As a result, gap(s) appears.

represents a major benefit as all gaps in the current schedule are stored in a separate list (*gap_list*) which speeds up the whole procedure [20]. This "gap-filling" approach is very useful as it significantly increases system utilization while respecting the start times of previously added jobs.

Often the schedule must be *updated* as events such as (early) job completions or machine failures appear [20]. In such a situation the `pbs_server` contacts the `pbs_sched` module which launches a time efficient schedule update procedure that updates the internal jobs' parameters in the *job_list* while recomputing the *gap_list* as well [20]. Commonly, job finishes earlier than expected as the schedule is built using processing time estimates which are typically overestimated. In such a case, the schedule is immediately *compressed*. The compression shifts jobs into earlier time slots that could have appeared as a result of an earlier job completion. During a compression, the relative ordering of job start times is kept [17]. This procedure is an analogy to the method used in Conservative backfilling [28].

Previous methods guarantee that the schedule will remain consistent with the current state of the system. Obviously, the schedule is also useful when *predicting the behavior* of the system. As the schedule holds all information concerning planned job execution it is typically used when an user's request concerning the status of his or hers jobs arrives to the `pbs_server` via the `qstat` command[3]. In such a case the `pbs_server` contacts the `pbs_sched` module which queries the schedule. Then, the information is returned to the user in a standard `qstat` response. An example of such a query is shown in Fig. 3 where the user can see the currently planned start time of his or her job.

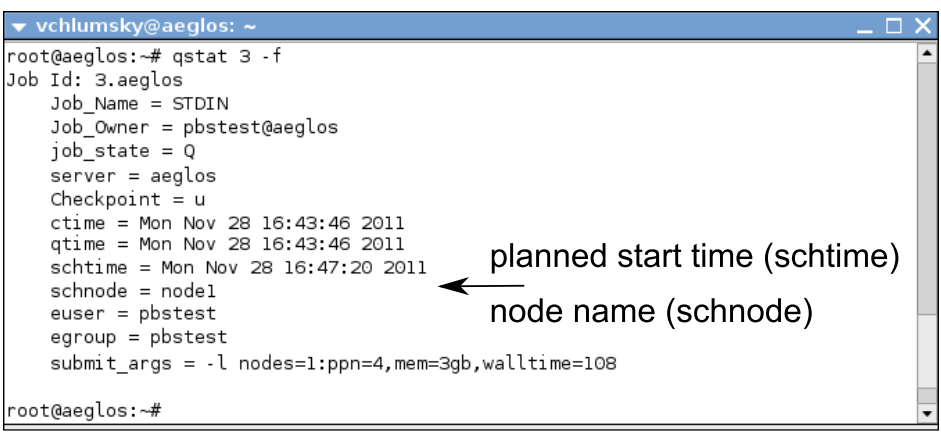

Figure 3. The output of the modified `qstat` command presenting the planned start time of a given job according to the current schedule

[3]In TORQUE, the `qstat` command is used to request the status of submitted jobs.

Last but not least, we have developed an *evaluation procedure* which analyzes an existing schedule with respect to several optimization criteria. Currently, we support makespan, slowdown, response time and wait time criteria [9]. The evaluation allows us to analyze the quality of constructed schedule. Moreover, it is used as a decision maker during a schedule optimization routine. For this purpose we use a Tabu Search-based *optimization metaheuristic* which we have proposed in our earlier work [21].

4. Experimental evaluation

The purpose of the experimental evaluation was twofold. First, we have measured the performance of a developed schedule-based solution with respect to the growing size of considered job scheduling problem, to see whether the complex data structure is capable of maintaining large schedules efficiently. Both runtime and memory requirements were of interest here. Second, we have analyzed whether the Tabu Search optimization procedure is able to improve the performance of the scheduler.

In the first experiment we have analyzed the time and memory needed when adding newly arriving jobs into the schedule. To analyze the influence of the growing size of the schedule the number of waiting jobs stored in the schedule has been subsequently increased and the effect on the runtime and memory consumption of the scheduler has been measured. We have started with an empty schedule and in a job-by-job fashion have increased its size up to 25,000 jobs. For each such setup, we have measured the time needed to add new job into the schedule and the amount of consumed RAM. The size of the Grid system was fixed according to the current status of the Czech NGI MetaCentrum [27] which includes 22 clusters having 219 nodes with 1494 CPUs. Data representing jobs were taken from the MetaCentrum workload log (July – November 2011). The main characteristics of jobs are depicted in Fig. 4. The jobs are rather heterogeneous in terms of duration and the level of parallelism. This workload log contained some 250,000 jobs that has been divided into 10 data sets. Each such a data set was then used to simulate the growing size of schedule (up to the 25,000 limit). The experimental installation of TORQUE was hosted on an Intel Pentium Dual Core E5700 3.0 GHz machine having 4 GB of RAM.

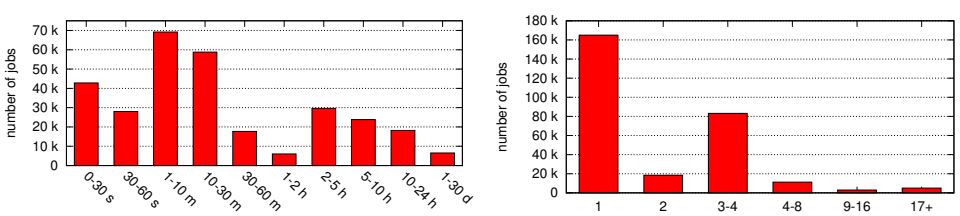

Figure 4. The distribution of job durations (left) and the distribution of job parallelism (right)

The results of the first experiment are shown in Fig. 5. The graph in the top left corner shows the average time (in microseconds (μs)) needed to add one job into an

existing schedule. It is the runtime of the "gap-filling" scheduling policy that finds suitable gap(s) for the incoming job. The size of such a schedule is depicted on the x-axis. As soon as the position is found and the job is added into the schedule both the *gap_list* and *job_list* must be updated. The runtime of the update procedure is shown in the top right corner of Fig. 5. The total time (sch. policy + update) is shown in the bottom left part of Fig. 5. The last graph (see Fig. 5 bottom right) shows the amount of used RAM as measured by the C `mallinfo` function[4]. This memory is allocated by the `malloc` during the construction of the schedule. The `malloc` function is used for allocating all schedule related structures. The experimental results indicate reasonable performance of the proposed solution as both a runtime and a memory consumption grows linearly with respect to the size of the schedule. Even for very large schedules the time needed to add a new job into an existing schedule is bellow 35 milliseconds, allowing us to solve frequent job arrivals without major delays.

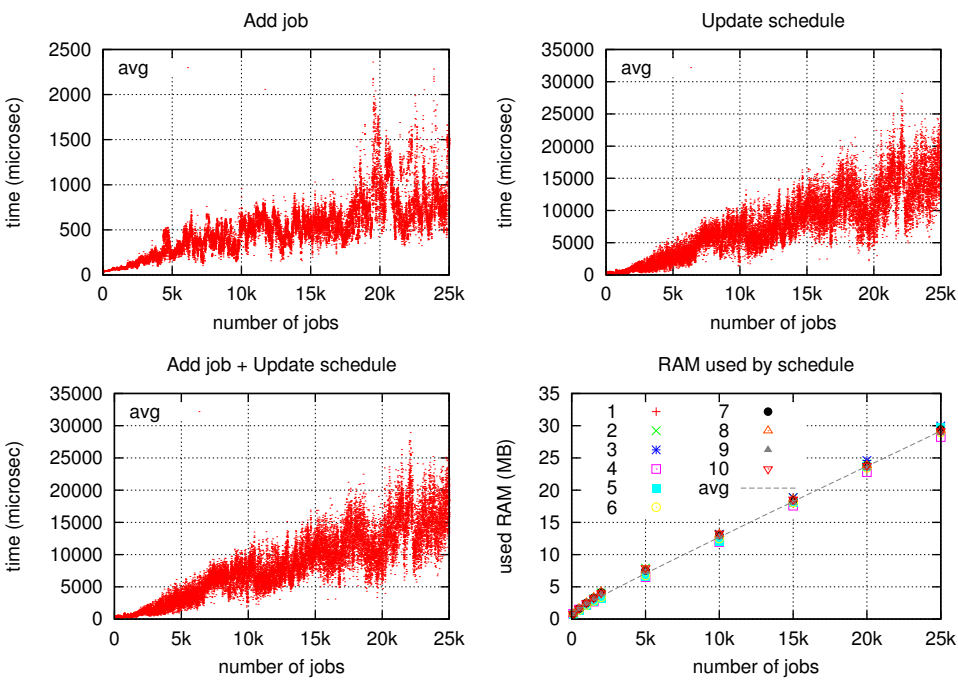

Figure 5. Runtime and memory requirements of the proposed schedule-based solution

In the second experiment the performance of applied Tabu Search (TS) optimization algorithm has been measured. This algorithm is designed to improve the quality of the initial schedule as generated by the "gap-filling" policy. TS is executed periodically every 5 minutes, and its runtime is currently bounded by 2 seconds. In order to

[4]Results for all 10 experiments are shown together with the average value.

allow comparison with other existing algorithms we have implemented our algorithm in a simulator since the TORQUE scheduler only supports FCFS and our conservative backfill-like scheduling policy. We have used Alea [18] simulator that contains implementations of popular algorithms such as FCFS, EASY backfilling (BF-EASY), Conservative backfilling (BF-CONS) or aggressive backfilling without reservations (BF). First, we have tested the simulator to see whether it generates reliable results. For this purpose, we ran an experiment where the same workload was loaded both into the TORQUE scheduler and into the Alea simulator[5]. Then, the results for FCFS and CONS, were compared between TORQUE and Alea. The results were almost the same, e.g., the difference in the avg. response time has been only 0.19% and 0.33% for FCFS and CONS respectively. Therefore, Alea has been found acceptable to perform reliable evaluation. Six different data sets from the Parallel Workloads Archive [8] have been used: MetaCentrum, HPC2N, KTH SP2, CTC SP2, SDSC SP2 and SDSC BLUE. We have chosen avg. wait time and the avg. slowdown criteria to be optimized simultaneously by the TS.

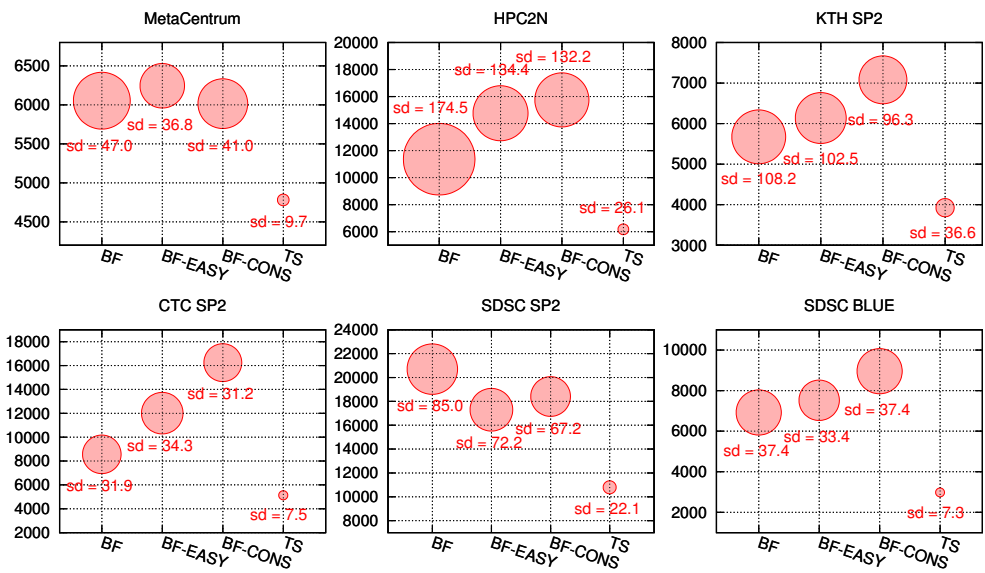

Figure 6. The comparison of average wait time and average slowdown as achieved by Tabu Search (TS) and remaining algorithms

The results of the second experiment are shown in Fig. 6. Here, the y-axis represents the avg. wait time while the diameter of the circle depicts the avg. slowdown. For better visibility FCFS is not presented in the figures as it produced very bad off-scale high results. The results show how evaluation and optimization can improve

[5]The experiment involved a sample from the HPC2N workload (jobs 187,306–188,307) that represented 6 days of job execution including a large peak of user activity.

the performance of the scheduler. Clearly, TS was able to outperform the remaining backfilling algorithms in all considered cases, decreasing both wait time and slowdown. These results support our idea that schedule-based solution accompanied by evaluation and optimization represents an advantageous scheduling approach.

The experiment presented above used precise processing time estimates. Naturally, when estimates are inaccurate, the performance of scheduling algorithms may get worse [33] and the schedule becomes inefficient unless a *schedule compression* is performed as discussed in Section 3.4. Moreover, the optimization metaheuristic, e.g., the Tabu Search can be efficiently used as a "recovery procedure" that helps to restore the good quality of the compressed schedule as we have shown in our recent works [19, 17].

5. Conclusion and future work

In this paper we have presented major extension to the widely used TORQUE Resource Manager. The extension replaces the original "naive" FCFS-like queue-based `pbs_sched` module with a complex schedule-based solution. Thanks to the applied schedule-based paradigm the solution supports planning, optimization and allows predictability. Users can now query the resource manager using standard tools such as `qstat` command to get up-to-date information concerning the expected start times of their jobs. Using it, they can better plan their work as the system's behavior becomes more predictable even for unskilled users. Also, the solution has been designed in order to allow for the application of advanced scheduling methods such as metaheuristics that would allow us to increase the quality of generated solutions with respect to selected optimization criteria. Initial synthetic tests presented in this paper as well as our earlier works [21, 17, 22] indicate that such optimization techniques allow us to achieve significantly better performance compared to the standard solutions such as EASY, Conservative backfilling, FCFS, etc. Typically, slowdown, response time and wait time can be improved once optimization is applied. Success is based on the fact that the use of a schedule (i.e., planning of future job execution) allows us to evaluate a constructed solution and improve it using local search-based optimization.

In the near future the developed scheduler will be thoroughly tested using real computer test-bed and real workloads with inaccurate runtime estimates. Last but not least, the software package will be made available to the public.

Acknowledgements

The access to the MetaCentrum computing facilities provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" LM2010005 funded by the Ministry of Education, Youth, and Sports of the Czech Republic is highly appreciated. The workload logs used in this paper were generously provided by the Parallel Workloads Archive.

References

[1] Abraham A., Buyya R., Nath B.: Nature's heuristics for scheduling jobs on computational Grids. [in:] *The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000)*, pp. 45–52, 2000.

[2] Abraham A., Liu H., Grosan C., Xhafa F.: Nature inspired meta-heuristics for Grid scheduling: Single and multi-objective optimization approaches. [in:] *Meta-heuristics for Scheduling in Distributed Computing Environments* [34], pp. 247–272.

[3] Adaptive Computing Enterprises, Inc. *Moab workload manager administrator's guide, version 6.1.4*, February 2012. `http://www.adaptivecomputing.com/resources/docs/`.

[4] Adaptive Computing Enterprises, Inc. *TORQUE Admininstrator Guide, version 3.0.3*, February 2012. `http://www.adaptivecomputing.com/resources/docs/`.

[5] Baptiste P., Pape C. L., Nuijten W.: *Constraint-based scheduling: Applying constraint programming to scheduling problems*. Kluwer, 2001.

[6] Baruah S., Funk S., Goossens J.: Robustness results concerning EDF scheduling upon uniform multiprocessors. *IEEE Transactions on Computers*, 52(9):1185–1195, 2003.

[7] Burke S., Campana S., Peris A. D., Donno F., Lorenzo P. M., Santinelli R., Sciaba A.: *gLite 3 user guide*. Worldwide LHC Computing Grid, January 2007.

[8] Feitelson D. G.: Parallel workloads archive (PWA), February 2012. `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[9] Feitelson D. G., Rudolph L., Schwiegelshohn U., Sevcik K. C., Wong P.: Theory and practice in parallel job scheduling. [in:] Feitelson D. G., Rudolph L., editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *LNCS*, pp. 1–34. Springer Verlag, 1997.

[10] Feitelson D. G., Weil A. M.: Utilization and predictability in scheduling the IBM SP2 with backfilling. [in:] *12th International Parallel Processing Symposium*, pages 542–546. IEEE, 1998.

[11] Gentzsch W.: Sun Grid Engine: towards creating a compute power Grid. [in:] *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 35–36, 2001.

[12] Hovestadt M., Kao O., Keller A., Streit A.: Scheduling in HPC resource management systems: Queuing vs. planning. [in:] *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *LNCS*, pp. 1–20. Springer, 2003.

[13] Huedo E., Montero R., Llorente I.: The GridWay framework for adaptive scheduling and execution on Grids. *Scalable Computing: Practice and Experience*, 6(3):1–8, 2005.

[14] Jones J. P.: *PBS Professional 7, administrator guide*. Altair, April 2005.

[15] Keller A., Reinefeld A.: Anatomy of a resource management system for HPC clusters. *Annual Review of Scalable Computing*, 3:1–31, 2001.

[16] Kleban S. D., Clearwater S. H.: Fair share on high performance computing systems: What does fair really mean? [in:] *Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, pp. 146–153. IEEE Computer Society, 2003.

[17] Klusáček D.: *Event-based Optimization of Schedules for Grid Jobs*. PhD thesis, Masaryk University, 2011.

[18] Klusáček D., Rudová H.: Alea 2 – job scheduling simulator. [in:] *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*. ICST, 2010.

[19] Klusáček D., Rudová H.: Handling inaccurate runtime estimates by event-based optimization. [in:] *Cracow Grid Workshop 2010 Abstracts (CGW'10)*, Cracow, Poland, 2010.

[20] Klusáček D., Rudová H.: Efficient data representation of large job schedules. In *Mathematical and Engineering Methods in Computer Science (MEMICS 2011) selected papers*, volume 7119 of *LNCS*. Springer, 2011. To appear.

[21] Klusáček D., Rudová H.: Efficient Grid scheduling through the incremental schedule-based approach. *Computational Intelligence: An International Journal*, 27(1):4–22, 2011.

[22] Klusáček D., Rudová H., Baraglia R., Pasquali M., Capannini G.: Comparison of multi-criteria scheduling techniques. [in:] *Grid Computing Achievements and Prospects*, pp. 173–184. Springer, 2008.

[23] Kurowski K., Back W., Dubitzky W., Gulyás L., Kampis G., Mamonski M., Szemes G., Swain M.: Complex system simulations with qoscosgrid. [in:] *Proceedings of the 9th International Conference on Computational Science: Part I*, volume 5544 of *LNCS*, pp. 387–396. Springer, 2009.

[24] Kurowski K., Oleksiak A., Piatek W., Weglarz J.: Hierarchical scheduling strategies for parallel tasks and advance reservations in grids. *Journal of Scheduling*, 11(14):1–20, 2011.

[25] María M. López E. H., Senar M. A.: Sensitivity analysis of workflow scheduling on Grid systems. *Scalable Computing: Practice and Experience*, 8(3):301–311, 2007.

[26] Matyska L., Ruda M., Šimon Tóth.: Work towards peer-to-peer scheduling architecture for the czech national grid. [in:] *Cracow Grid Workshop (CGW 2010)*. Cyfronet AGH, 2010.

[27] MetaCentrum, February 2012. `http://www.metacentrum.cz/`.

[28] Mu'alem A. W., Feitelson D. G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.

[29] Skovira J., Chan W., Zhou H., Lifka D.: The EASY – LoadLeveler API project. [in:] Feitelson D. G., Rudolph L., editors, *Job Scheduling Strategies for Parallel Processing*, volume 1162 of *LNCS*, pp. 41–47. Springer, 1996.

[30] Smith W., Foster I., Taylor V.: Scheduling with advanced reservations. [in:] *In-*

ternational Parallel and Distributed Processing Symposium (IPDPS '00), pp. 127–132, 2000.

[31] Süß W., Jakob W., Quinte A., Stucky K.-U.: GORBA: A global optimising resource broker embedded in a Grid resource management system. [in:] *International Conference on Parallel and Distributed Computing Systems, PDCS 2005*, pp. 19–24. IASTED/ACTA Press, 2005.

[32] Thain D., Tannenbaum T., Livny M.: Condor and the Grid. [in:] Berman F., Fox G., Hey T., editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., 2002.

[33] Tsafrir D.: Using inaccurate estimates accurately. [in:] Frachtenberg E., Schwiegelshohn U., editors, *Job Scheduling Strategies for Parallel Processing*, volume 6253 of *LNCS*, pages 208–221. Springer Verlag, 2010.

[34] Xhafa F., Abraham A.: *Metaheuristics for Scheduling in Distributed Computing Environments*, volume 146 of *Studies in Computational Intelligence*. Springer, 2008.

[35] Xhafa F., Abraham A.: Computational models and heuristic methods for Grid scheduling problems. *Future Generation Computer Systems*, 26(4):608–621, 2010.

[36] Xu M. Q.: Effective metacomputing using LSF multicluster. [in:] *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pp. 100–105. IEEE, 2001.

Affiliations

Václav Chlumský
CESNET z.s.p.o., Prague, Czech Republic, `vchlumsky@cesnet.cz`

Dalibor Klusáček
CESNET z.s.p.o., Prague, Czech Republic, `klusacek@cesnet.cz`

Miroslav Ruda
CESNET z.s.p.o., Prague, Czech Republic, `ruda@cesnet.cz`

Received: 9.12.2011
Revised: 14.02.2012
Accepted: 23.04.2012