A highly configurable and efficient simulator for job schedulers on supercomputers

Carsten Karbach

Institute for Advanced Simulation
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH
c.karbach@fz-juelich.de

Abstract: The supercomputers maintained by the Jülich Supercomputing Centre (JSC) are run in batch mode and are shared among all active users. Jobs are submitted to the job scheduler, which determines for each job the time of execution. It manages running and waiting jobs, sorts them by a given priority and executes the jobs by fitting them into the current system state. For users it is often difficult to comprehend the job scheduler's algorithm. Thus, a prediction of the future system allocation of currently running and queued jobs is valuable. It helps users to plan job submissions and supports administrators by optimising the system load.

This paper summarises the design and implementation of a configurable simulation for various job schedulers. The developed simulation program called JuFo focuses on the job schedulers Loadleveler and Moab as these are the most important batch systems for the supercomputers provided by the JSC. A major design goal is to keep the simulation independent from special job schedulers, which is achieved by the generic configuration of JuFo. Finally, a test framework is developed, which allows for evaluating the accuracy of the generated schedules.

1 Introduction

Computing time granted on supercomputers like those provided by the JSC is a rare resource. On the one hand, the usage of this resource can be optimised by improving the performance of the users' algorithms executed on the supercomputers. On the other hand, administrators of the supercomputers have various options for minimising idling times of the compute resources. The choice and configuration of the operating system for the supercomputers strongly influences their efficiency. Especially the job scheduler, which "receives a stream of job submission data and produces a valid schedule" [KSY99], represents a promising supercomputer component allowing for significant throughput gain. In order to enhance the scheduler's efficiency for instance the scheduling algorithm or various types of configuration parameters can be adapted. These changes are risky, since their effects mostly can be tested only on the production system. As a result, a simulation program for job schedulers is required. The scope, architecture and design of the developed simulation program called JuFo are outlined in this paper, while a detailed documentation of its development is given by [Kar12].

Besides supporting the supercomputer administration, JuFo is also used for on-line prediction of the future schedule. Based on current job requests and available compute resources the simulation program estimates the dispatch time for submitted jobs. Thus, it extracts the dispatch times of currently waiting user jobs as well as future idling resources available for new appropriately sized job requests. Users obtain a better understanding of the scheduling algorithm, which allows for optimising the system load and the usage of computing time. In order to apply JuFo as on-line prediction tool the simulation time has to be limited to a few minutes, which demands a very efficient implementation.

2 Problem definition

Implementations of many scheduling algorithms used by today's job schedulers are proprietary and are not published. There are merely rough documentations about their general functionality. This prevents users of job schedulers from understanding the underlying scheduling algorithm. Not many job schedulers provide a simulation mode, which would allow for predictions of the future schedule. Therefore, a customisable job scheduler simulation is required.

The developed simulation program is characterised by its high number of configuration parameters, its extensibility and its simple integration into the monitoring system LLview (see [Cen05]). JuFo aims to be highly configurable in order to apply it as simulator of various batch systems. It is based on a detailed analysis of the scheduling systems Moab Workload Manager [Com12] and IBM Tivoli Workload Scheduler LoadLeveler [IBM09]. Both job schedulers are used on JSC supercomputers, which allows for comparing the simulation's results with real-world schedules. The scheduling algorithms and configuration options of these batch systems are steadily advanced. As a result, JuFo's functionality will have to be extended and adapted accordingly. JuFo represents an extensible basis for simulating the analysed job schedulers. The simulation program can also be extended by adding new scheduling algorithms or site specific configuration parameters. Moreover, JuFo is integrated into LLview's monitoring workflow. LLview is a generic monitoring system for supercomputers. It collects real-time information of the monitored system and visualises it on a single well-arranged display. Based on collected status data JuFo simulates the future job schedule, which is forwarded to LLview's visualisation client. Therefore, JuFo exclusively focuses on the job scheduler simulation. Input data as well as visualisation of the simulation's results are provided by the monitoring environment.

2.1 Related work

In this subsection related applications for the prediction of job schedulers are outlined. The idea for JuFo arises from an existing LLview module for schedule prediction named *SchedSim* (see [Cen05]). SchedSim is an advanced simulator for the Blue Gene architectures maintained at JSC. But its efficiency is not sufficient for large clusters running

thousands of jobs. Moreover, it is based on an implicitly defined XML data format, which complicates future extensions. In [SW02] a prediction application for job execution time and queue time is presented. Based on historical data similar job requests are extracted, which are used to derive the most likely execution time for each job. Using this data the scheduling algorithm FCFS, which is explained in section 4, is implemented for predicting the job queue time for all waiting jobs. This prediction program is implemented only for the batch system PBS (see [Eng13] for PBS documentation). An alternative approach is the prediction of job wait times exclusively based on historical data without any assumptions on the underlying job scheduler. This solution is used in [NBW08]. It simplifies the configuration of the scheduler prediction, since no knowledge of the scheduler is required. However, it relies on the existence of log files about past workload traces. Finally, for some schedulers specific prediction applications are developed such as the *showstart* command for Moab [Com12, p.306] or the simulation mode wrapper for the batch system SLURM [Luc11]. In comparison with the presented applications JuFo uniquely provides an efficient code basis for simulating job dispatch times for all currently submitted jobs based on knowledge of the actual scheduling algorithm. It is highly extensible and directly integrated into a supercomputer monitoring architecture, which visualises the simulation results immediately. Not only the start time for a single job request is predicted, but the entire future schedule is simulated. Furthermore, no historical status data is required. JuFo can generate accurate schedules right from the launch of a new supercomputer. The only requirement is to map the underlying scheduling algorithm and policies to JuFo's configuration. To sum up, JuFo is designed as efficient replacement for SchedSim in LLview allowing extensibility and at the same time providing a similar set of configuration parameters, which define the scheduling algorithm.

2.2 Job scheduler definition

From computer theory job schedulers consist of a hierarchy of three independently working scheduler tasks: the global job scheduler, which is responsible for mapping complete jobs to compute resources, the task scheduler, which implements multitasking on a single processor and the thread scheduler, which represents the finest scheduling system and decides, which thread of a process is executed at a specific time frame (see [Arn97]). JuFo only simulates the global scheduler layer. A global job scheduler allocates available compute resources such as compute nodes, cores and memory to jobs requesting these resources. This task is divided into prioritising job submissions, placing jobs into a schedule generated by the underlying scheduling algorithm and managing the system's compute resources. Moreover, the scheduler needs to consider reservations, job dependencies and constraints given by the system's waiting queues.

2.3 The scheduling problem

From a more abstract perspective a job scheduler approaches a special case of the *resource-constrained project scheduling problem (RCPSP)* [BK12]. It defines a combinatorial optimisation problem of a schedule regarding an objective function. A schedule is modelled as vector, where each component determines the start date of a so called *activity*. By applying the RCPSP to job schedulers an activity is synonymous with job. The objective function evaluates the quality of a schedule. E.g. a simple objective function is the time duration covered by a schedule to run all jobs. The RCPSP searches the schedule, which optimises the objective function. This problem is proven to be *NP-hard* [BK12, p. 34]. Thus, heuristic approximations are used by the analysed scheduling systems, which have to be simulated by JuFo. Note, that many scheduling algorithms require knowledge of the job duration times in order to generate schedules. Since JuFo is used as on-line prediction, the actual runtime of a job is unknown. Instead, the wall clock limit provided by the users as upper limit for the runtime is used as runtime within the simulation. The accuracy of the wall clock limit is crucial for the correctness of the simulation result.

2.4 Limitations

Despite the aim of covering a wide range of supercomputers, JuFo cannot be applied for all systems. Since batch processing is the predominant job placement strategy for supercomputers, JuFo is not designed for the simulation of time-shared resources, where jobs are allocated to the processors via time slicing. Moreover, the concepts of job preemption and migration are also not considered. Jobs are expected to be requesting compute nodes and a number of processors per node and they are grouped into waiting queues, which each define special scheduling policies. These assumptions as well as the scheduling algorithms supported by JuFo are derived from the analysis of present supercomputers. Although JuFo provides a large number of configuration parameters, it cannot replace batch system specific simulation applications developed for testing the impact of changing details of the batch system's configuration. Instead JuFo is regarded as simplified model for the simulated batch systems. Thereby, it needs to balance the conflicting targets of generalization, efficiency, flexible configuration and extensibility.

3 Monitoring environment

JuFo is embedded into the monitoring environment provided by LLview. It allows to retrieve the status data needed as input for JuFo for a large number of target systems. At the same time LLview is capable of visualising JuFo's simulation outcome. As a result, the simulation's objective is narrowed down to the generic simulation of a global job scheduler without additional effort for data retrieval or visualisation. Therefore, JuFo is integrated into LLview's workflow as a stand-alone module.

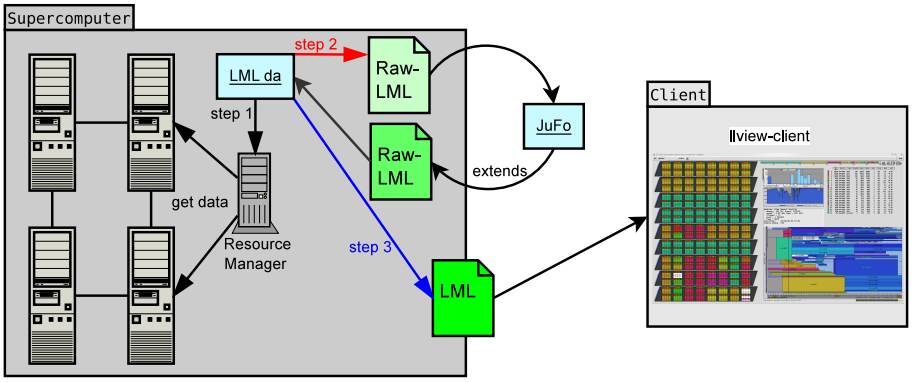

Figure 1: LLview's monitoring environment

The architecture of LLview's monitoring environment is depicted by figure 1. LLview is split into LML_da and the client application. While the former is executed on the monitored supercomputer and collects current status data such as running jobs or the availability of the compute nodes, the latter visualises the collected status data. Usually LML_da updates the status data every minute by triggering the particular resource manager for the current state. The Large-scale system Markup Language (LML) works as data format and abstraction layer between both components of LLview (see [WFK+11]). LML_da is composed of a set of independent modules, which process LML files. There is one module for parsing job and node data from the output of corresponding batch system commands labelled with *step 1*. A second module merges the outcome of the different batch system commands into a single raw LML file, which is marked with *step 2*. A raw LML file comprises a rather flat structure of job and resource data, which is not easily visualised. Therefore, another module converts the raw data into an LML file close to its actual visualisation in *step 3*. This file is transferred via SSH or HTTP to the client application, which parses the final LML file and renders it. JuFo is integrated into this workflow in the same manner as the described modules of LML_da. The raw LML file of the second step is provided as input file containing currently running and submitted jobs as well as the state of the compute nodes and the requested resources of the jobs. Based on that data the global job scheduler is simulated with JuFo resulting in a predicted start date for each submitted job, which is added to the output LML file. Afterwards, this output file is parsed by further LML_da modules and converted into corresponding visualisation data. Embedding JuFo into this architecture is a powerful solution in order to provide a generic status data description independent of the particular batch system, which is abstracted in LML.

4 Job scheduling

First-Come-First-Served (FCFS), List-Scheduling and Backfilling (see [KSY99]) are established scheduling algorithms used by LoadLeveler and Moab to solve the described scheduling problem. FCFS sorts jobs by their submission date and simply runs them in this order as soon as possible. An example schedule generated by FCFS is depicted in figure 2. Five cores are shared among six jobs. The job names indicate their ranking regarding their submission dates with *j1* as highest ranked job. List-Scheduling sorts jobs

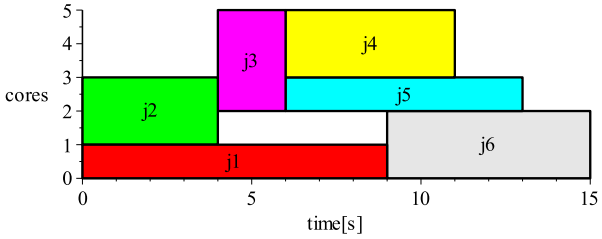

Figure 2: Job schedule generated by FCFS

by an arbitrary priority formula. The job list is traversed starting with the highest ranked job. The first job, whose requested compute resources are available at the current time, is dispatched. Therefore, in contrast to FCFS List-Scheduling allows to start lower ranked jobs with less resource requirements before higher ranked jobs. For the above example schedule List-Scheduling would dispatch *j4* at the schedule's beginning and delay the start of the higher ranked job *j3*. While FCFS guarantees fairness, List-Scheduling tends to produce higher average throughput. Backfilling combines the advantages of FCFS and List-Scheduling: if the highest prioritised job, also known as *top dog*, cannot be started due to insufficient available resources, the requested amount of resources is reserved for this job as soon as they are released by the completion of another job. Available resources before the top dog's start can be used by lower ranked jobs as long as their submission does not collide with the top dog reservation. Backfilling generates predominantly fair schedules and simultaneously uses resources, which would have been idling when applying FCFS. These scheduling algorithms are implemented by JuFo. They represent a basis for novel scheduling algorithms and allow for simulating both analysed job schedulers LoadLeveler and Moab with various configuration parameters.

5 Simulation's design

Derived from the analysis of common job schedulers JuFo's main components are formed by the resource manager, scheduling algorithm and job prioritisation. A resource manager tracks available compute resources and determines, whether jobs can be dispatched on currently available resources. The scheduling algorithm implements the placement of

jobs into the schedule, while the job prioritisation component sorts waiting jobs according to a given priority formula. Moreover, a timeline is implemented for managing scheduling events such as job submissions, completions or resource state changes. These main packages of JuFo are depicted in figure 3.

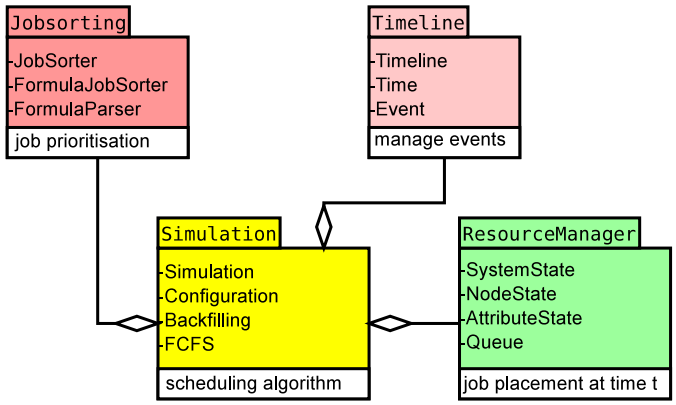

Figure 3: Package overview of JuFo's architecture, each package lists its key classes and contains a short description of its overall task

Besides the described main components, JuFo comprises packages for handling LML files and optimising the resource management (see [Kar12]). The *Simulation* package coordinates the interactions of all components by implementing the scheduling algorithms described in section 4. It parses the provided configuration and chooses the type of resource manager and job sorting algorithm. Note, that the packages can only interact via abstract classes, so that the concrete implementation of each package can be easily exchanged. Jobs are ranked by the *Jobsorting* package. An arbitrary mathematical term, which is derived from the configured priority formula, is evaluated dynamically for each job. This formula amongst others can contain the queue time, the number of currently active jobs, the number of requested compute nodes or the job's wall clock limit. The corresponding attributes are retrieved from the input LML file. The timeline stores a list of relevant scheduling events. E.g. for each job dispatch or completion a new event is added. An event indicates that the state of the simulated system has changed. A job dispatch for instance consumes available compute nodes, while a job completion releases used resources. In general, the scheduling algorithm traverses the timeline events. The resource changes defined by each event are applied via the resource manager. Afterwards, the resource manager is used to check, whether any waiting job can be dispatched on the adapted resources. If jobs can be dispatched, corresponding events are inserted into the timeline. Then the scheduling algorithm continues with the next timeline iteration. The *ResourceManager* package implements, whether and how jobs can be placed according to currently available compute resources. It tracks the number of available compute nodes, the number of active jobs for each user and the number of jobs for each queue. Based on this data it decides, whether a job is allowed to be dispatched and allocates the requested compute resources. Here, many scheduling rules have to be considered. E.g. nodes can be reserved for a special

user group, there are upper limits for active jobs per user, compute nodes can be shared by multiple jobs or can only be used by one job at once. Moreover, jobs do not only require CPUs, but also memory, software licenses and certain network topologies. The availability of these resources varies over the simulated time span and has to be traced by the resource manager.

In order to give an idea on how JuFo's components interact, figure 4 depicts the implementation of the scheduling algorithm FCFS. This function is located in the Simulation package and a list of waiting jobs is passed as argument.

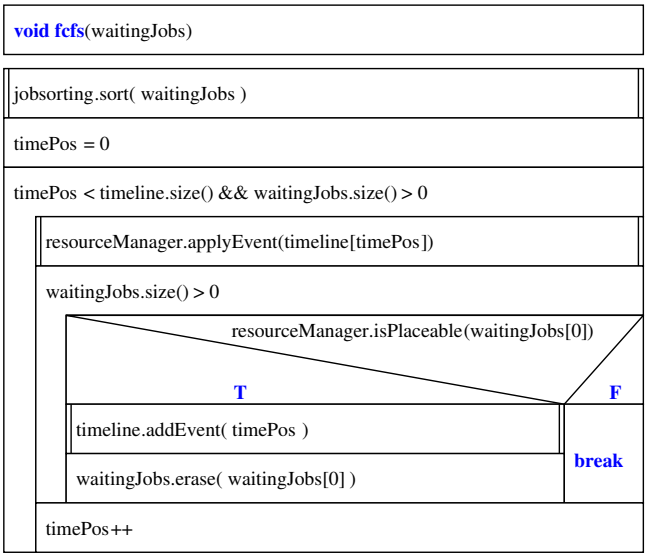

Figure 4: Pseudo code implementation of FCFS

At first the jobs are sorted by calling the jobsorting instance. In this example the sort criterion is expected to be static, so that a single call of the sorting procedure is sufficient. Afterwards, the timeline is traversed as long as there are unplaced waiting jobs. The changes of the current event are applied with the resource manager. The inner loop searches for jobs, which can be placed into the current system state. If an eligible job is found, it is inserted and a corresponding event for the job completion is created and sorted into the timeline. As soon as the currently highest ranked waiting job cannot be placed, the inner loop is aborted and the next event of the timeline is conducted. This is an example for a simple scheduling algorithm. Advanced scheduling algorithms such as Backfilling and List-Scheduling are also implemented in JuFo (see [Kar12]).

6 Verification framework

As explained in section 5 the simulator is formed by a set of encapsulated modules, which can be tested separately with the help of unit tests. These tests are designed to validate the correctness of independent software modules by comparing their actual behaviour with the expected behaviour (see [IEE86]). More than 100 test cases are developed for testing the different JuFo components. These tests can be used as regression tests for future development and they function as a set of example applications showcasing how to use and extend JuFo. But unit tests are mostly based on synthetic input data, because their results must be known in advance for evaluating, whether the tested module works as expected. In order to apply JuFo as on-line prediction of a real supercomputer the accuracy of its results needs to be evaluated with real-world input data. Since most of the simulated scheduling algorithms are only roughly documented, the schedules predicted by JuFo have to be compared with real schedules of the target system. As a result, a test framework is developed for the verification of JuFo's results. It allows for gathering the actual workload of the particular supercomputer. The simulation is run based on the status data of the past. Afterwards, the predicted schedule is compared with the actual schedule produced by the real scheduler. With the help of this test framework the scheduling system of the JSC supercomputer JUROPA (see [Cen13]) is successfully simulated by JuFo.

The verification procedure is composed of the following steps:

1. Gather LML snapshots of the system status every minute throughout the test period
2. Combine all snapshots into a single LML file
   (a) Determine the exact wall clock limits of jobs, which finish in the test period
   (b) Store the actual start time for each job
   (c) Mark jobs, which start in the test period, as *waiting* and jobs, which started before the test period, as *running*
   (d) Detect jobs submitted within the test period and mark them as future jobs
   (e) Mark jobs accordingly, which are removed from the queue before being started
3. Run JuFo with the generated test file and simulate the schedule
4. Compare this schedule with the real schedule traced in step 1

This algorithm focuses on the accuracy of the simulation program by avoiding unpredictable determinants such as imprecise wall clock estimates, early job cancellations or highly prioritised future job submissions. These determinants cannot be extinguished when using JuFo as on-line prediction. But they need to be suppressed in order to allow a clear view on how accurately JuFo simulates the real scheduler.

6.1 Test results for JUROPA

The presented verification procedure is applied to the supercomputer JUROPA. Tests were run on eight days from July till August, 2012. The test duration varies between about two and six hours. At each test date JuFo simulated the start dates for 107 up to 280 jobs. Table 1 lists the key measures of the error distributions of each verification run. Errors are calculated as the difference of the job's real start date and the predicted start date provided by JuFo. Columns four to eight hold the mean, median, minimum, maximum and standard deviation of the sample. All of these measures are listed in seconds. Note, that this table only contains those jobs, which started within the test period and which also were predicted to start in the test period. This ensures that the actual start date is known and that only jobs within the test period are considered.

Test date	**Duration**	**Jobs**	**Mean[s]**	**Median[s]**	**Min.[s]**	**Max.[s]**	σ**[s]**
07/23/2012	01:43:27	107	33.6	34	-3020	4871	1154.3
07/24/2012	03:50:10	226	-38.3	31	-8175	7340	1709.6
07/25/2012	02:12:01	186	-773.6	-116	-7467	3336	1851.0
07/27/2012	04:22:20	191	34.2	42	-11038	7515	1866.3
08/01/2012	02:35:14	122	642.6	76	-4637	6815	1886.8
08/02/2012	04:25:37	280	468.6	64	-10462	14620	3360.9
08/06/2012	03:19:02	133	-326.1	54	-7783	7516	2473.2
08/07/2012	05:39:31	249	289.3	49	-15378	19796	4104.9

Table 1: prediction error distributions for example tests on JUROPA

The mean values range from about -13 to 11 minutes. Considering job run times, which vary between 30 minutes and 24 hours, these errors are acceptable. Along with the median values between -2 and 1.27 minutes the results show, that on average JuFo provides a reasonable simulation for JUROPA's job scheduler.

However, the minimum, maximum and standard deviation columns indicate inaccurately placed jobs. There are outliers for each test run, which also account for the large standard deviations. The main reasons for misplaced jobs are wrong sorting of jobs, scheduling rules manually adjusted by the system administrators and missing reservations. Some users submit many identical job requests at the same time, which leads to identical system priorities. Without access to the actual implementation of the job scheduler, it is hard to discover how jobs with identical system priorities are ranked. Not every scheduling rule can be obtained dynamically by JuFo. As a result, every change of the scheduler configuration has to be propagated to JuFo in order to keep its accuracy as high as possible. Finally, in case of JUROPA it is not permitted for a normal user to list currently active reservations. That is why the above simulations were run without knowledge of reservations, which might influence the generated schedule. However, these shortcomings can be solved by running JuFo from a more privileged account, where for instance reservations and current scheduling rules can be queried. Furthermore, this verification framework can be used for iterative improvement of JuFo's configuration in order to reconstruct the behaviour of the actual job scheduler as good as possible. In the same manner this veri-

fication process can be conducted for other supercomputers. The key determinants of the scheduler's behaviour have to be extracted from its documentation. Starting with that configuration JuFo could be refined based on a similar set of sample runs until its accuracy is acceptable for a valid on-line prediction.

7 Conclusion

This paper presents JuFo, a simulator for global job schedulers of supercomputers. It is designed for on-line prediction of future job schedules and represents an efficient code basis for the simulation of a common range of scheduling algorithms. JuFo is embedded into the monitoring environment of LLview, which handles data retrieval and visualisation of the simulation results. The data format LML functions as communication interface for the different modules of LLview and JuFo. Supported scheduling algorithms are FCFS, List-Scheduling and Backfilling, which can be configured by a large number of parameters allowing for a detailed reconstruction of the particular job scheduler. The key components of the implementation are resource manager, job sorting algorithm, timeline and the scheduling algorithm. These components are strictly encapsulated, since they can only interact via interfaces, so that their concrete implementation can be easily exchanged. This concept also simplifies the developed unit tests, which can focus on each of the independent components. Additionally, a verification framework is implemented for comparing the job scheduler model of JuFo with the real job scheduler algorithm. This framework is successfully applied for improving and evaluating the accuracy of the schedule prediction for the supercomputer JUROPA.

In a similar manner JuFo is planned to be established as on-line prediction for other supercomputers. New challenges are the assignment of further types of resources such as accelerators, the growing number of job schedulers and configuration parameters and the increase of submitted jobs, which emphasises the demand for a highly efficient implementation. At the same time the accuracy of the prediction results has to be steadily improved. A promising approach is enhancing the simulation with speculative data derived from past workload traces. Instead of using the wall-clock limit provided by the users, an additional module can be integrated into LLview's workflow providing more accurate predictions of the actual job duration. Again, the presented verification framework can be used for iterative improvement of JuFo's reliability and accuracy.

References

[Arn97] Alfred Arnold. *Untersuchung von Scheduling-Algorithmen für massiv-parallele Systeme mit virtuell gemeinsamem Speicher*. PhD thesis, Jülich Research Centre, ZAM, 1997.

[BK12] Peter Brucker and Sigrid Knust. *Complex Scheduling*. Springer Verlag, 2012.

[Cen05] Jülich Research Centre. LLVIEW: graphical monitoring of LoadLeveler controlled cluster. http://www.fz-juelich.de/jsc/llview/, March 2005.

[Cen13] Jülich Research Centre. JUROPA / HPC-FF System Configuration. http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUROPA/Configuration/Configuration_node.html, June 2013.

[Com12] Adaptive Computing. *Moab Workload Manager, Administrator Guide*. http://www.adaptivecomputing.com/resources/docs/, 7.0 edition, 2012.

[Eng13] Altair Engineering. The Portable Batch System. http://www.pbsworks.com, June 2013.

[IBM09] IBM. *Tivoli Workload Scheduler LoadLeveler, Using and Administering*. IBM, http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.loadl.doc/llbooks.html, 3.5 edition, January 2009.

[IEE86] IEEE Standards Board. IEEE Standard for Software Unit Testing: An American National Standard, ANSI/IEEE Std 1008-1987. Technical report, The Institute of Electrical and Electronics Engineers, 1986.

[Kar12] Carsten Karbach. Design and implementation of a highly configurable and efficient simulator for job schedulers on supercomputers. Master's thesis, FH-Aachen, University of Applied Sciences, http://hdl.handle.net/2128/4703, August 2012.

[KSY99] Jochen Krallmann, Uwe Schwiegelshohn, and Ramin Yahyapour. On the Design and Evaluation of Job Scheduling Algorithms. http://www.gwdg.de/fileadmin/inhaltsbilder/Pdf/Yayhapour/cei_ipps99.pdf, 1999.

[Luc11] Alejandro Lucero. Simulation of batch scheduling using real production-ready software tools. http://www.bsc.es/media/4856.pdf, 2011.

[NBW08] Daniel Nurmi, John Brevik, and Rich Wolski. QBETS: Queue Bounds Estimation from Time Series. In Eitan Frachtenberg and Uwe Schwigelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 76–101. Springer-Verlag, 2008.

[SW02] Warren Smith and Parkson Wong. Resource Selection Using Execution and Queue Wait Time Predictions. Technical Report NAS-02-003, Computer Sciences Corporation, NASA Ames Research Center, Moffett Field, CA 94035, 2002.

[WFK+11] Gregory R. Watson, Wolfgang Frings, Claudia Knobloch, Carsten Karbach, and Albert L. Rossi. Scalable Control and Monitoring of Supercomputer Applications using an Integrated Tool Framework, September 2011.