Procedia Computer Science

Volume 66, 2015, Pages 661–669

YSC 2015. 4th International Young Scientists Conference on Computational Science

Introducing new backfill-based scheduler for SLURM resource manager

Sergei Leonenkov and Sergey Zhumatiy[1]

[1]*Research Computing Center, Lomonosov Moscow State University, Russia*

{leonenkov, serg}@parallel.ru

Abstract

The work proposes a design for a new external scheduler for SLURM (Simple Linux Utility for Resource Management). Schedulers, included in SLURM by default are good enough for many sites, but big supercomputers serving many users meet limitations of standard SLURM schedulers. In this work we discover methods to break these limitations via implementing new portable SLURM scheduler. We address the problem of maximizing the number of users, whose requests are processed in each given moment of time, and decrease start time of user's first task. Our approach is based on standard backfill algorithm and includes additional features, such as simplification of SLURM priority system, replacing slow SQL-based accounting checks by faster ACL checks and upgrading cluster administration convenience.

Keywords: SLURM; supercomputer; scheduling algorithms; backfill

1. Introduction

Every year both the number of active users and the computational capabilities of supercomputers all around the world are rapidly growing. The demand for such systems is increasing in a wide range of different applications, including scientific computations, solving various industrial and financial tasks. The users of high performance clusters have different access permissions, they can request different numbers of processors for solving their tasks, starting with a single processor and up to several thousands of them. The execution time for these tasks can range from a couple of seconds to several hours or even days. Effective processing of all requests in real time is a non-trivial task and it is a main challenge for high-performance systems' administrators. Lomonosov supercomputer is used for the research conducted by the scientists and students of the Lomonosov Moscow State University; currently it holds the second place in the Top50 list of supercomputers in CIS with the best performance measured by the Linpack test [5]. Also it is on the 58th place in a well-known international rating of the highest performing supercomputers called Top500 (as per November 18, 2014) [7].

Selection and peer-review under responsibility of the Scientific Programme Committee of YSC 2015

doi:10.1016/j.procs.2015.11.075

Lomonosov supercomputer consists of 12,422 CPUs, but even this amount is not enough to provide all of MSU research groups with necessary computation power. All CPUs are physically separated into groups of 4 or 6, called nodes. A selection of nodes may be combined into a logical group, which is called *partition* and includes a queue of incoming jobs. Partitions can be limited by, for example, indicating users who can use them, size of a job or processing time limits. Each partition focuses on the specific needs of users. Current configuration of the Lomonosov supercomputer includes 8 separate sections, each consisting of 1 to 4096 nodes [3-4]. As a rule 200-400 jobs are processed every day and requested by 30-50 different users in general. There are approximately 1000 active accounts and this number is still growing every year. To cope with the constantly growing workload a SLURM (Simple Linux Utility for Resource Management) system is used. This high-workload state of Lomonosov supercomputer and deficit of CPU-hours for lots of science groups faces Research Computing Center developers with problem of resource management software optimization. SLURM solution fits best for Lomonosov supercomputer administration purposes, but lack of abilities to easily add new features to manage job queues forces us to create our own external scheduling solution.

2. Background and Related Works

SLURM is a highly scalable, fault-tolerant cluster resource manager and job scheduler for big computational systems. Lawrence Livermore National Laboratory started developing SLURM to manage resources of their own supercomputers in 2001. SLURM cluster manager is used on many supercomputers specified in Top500 rating, including half of supercomputers stated in the top 10 (as per November 2013). Source code is written in C programming language. The system is available under GNU GPL V2 license and is well-documented. Two major objectives set by the developers of SLURM were high scalability and portability [1, 10].

SLURM is based on the hierarchical model of supercomputer management systems. The central agent of the manager is a main controller, which contains a slurmctld managing daemon. In order to improve fault-tolerance of the manager, the main controller could be backed up by a spare controller in certain architectures. Slurmctld's major purpose is to commit resources for jobs which were set by users [6].

Each separate node has slurmd daemon for managing. These daemons start and control a job on the node, namely they receive a job from the main controller and start it directly on cores; also they monitoring job state [6].

User commands include: sacct, salloc, sattach, sbatch, sbcast, scancel, scontrol, sinfo, smap, squeue, srun, strigger and sview. All these commands could be run both from the controlling server and cluster nodes [2].

SLURM is designed for heterogeneous clusters with up to 10 million processors possible. It is successfully used on a supercomputer with more than 98.000 nodes. Those who use a supercomputer managed via SLURM can set up to 1000 jobs for execution per second. Manager can perform up to 500 jobs per second (depending on the system configuration and equipment).

System administrator can set logical configuration of computing system, which would be supported by SLURM, flexibly and vary a set of cluster parameters easily by changing a configuration file or using appropriate commands.

2.1 Backfill algorithm

One of the key benefits of SLURM is modularity; dozens of additional plugins are available. Schedulers are also part of SLURM standard configuration, one of them (sched/backfill) is used for optimization of Lomonosov supercomputer operation. Lomonosov supercomputer uses a standard

scheduler and backfill algorithm. This scheduling algorithm is being used on a great deal of clusters and considered effective. A research showed that this algorithm allows to increase density of supercomputer resources' use by 20% and decrease average waiting time for setting jobs for execution [8].

Table 1. Backfill algorithm pseudocode.

1. Find the shadow time and extra nodes
    1. Sort the list of running jobs according to their expected termination time
    2. Loop over the list and collect nodes until the number of available nodes is sufficient for the first job in the queue
    3. The time at which this happens is the *shadow time*
    4. If at this time more nodes are available than needed by the first queued job, the ones left over are the extra nodes
2. Find a backfill job
    1. Loop on the list of queued jobs in order of arrival
    2. For each one, check whether either of the following conditions hold:
        - It requires no more than the currently free nodes, and will terminate by the shadow time, or
        - It requires no more than the minimum of the currently free nodes and the extra nodes
    3. The first such job can be used for backfilling

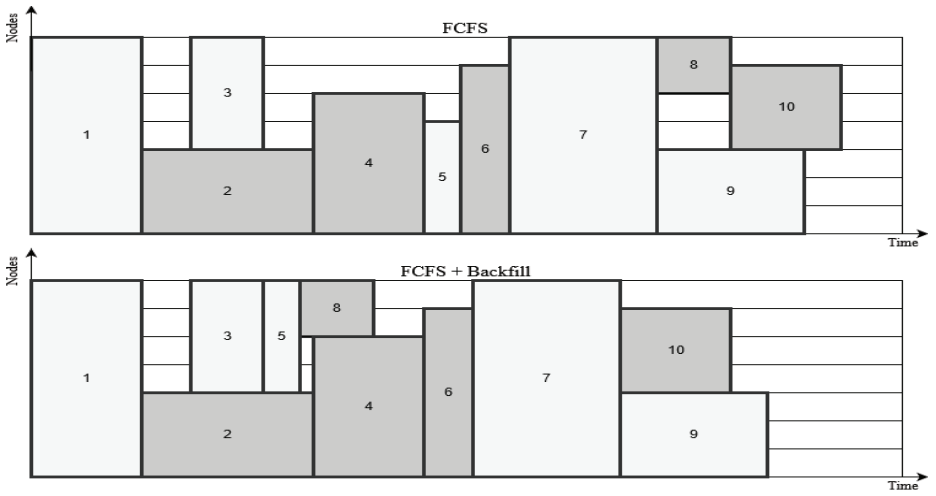

Fig. 1. Examples of FCFS and FCFS + Backfill.

Backfill algorithm is based on the First Come First Served principle. This means that jobs start in the same order they were put in a queue (if resources for each specific job execution are available). While Backfill algorithm (pseudocode is shown on Table 1) is running, a queue of jobs waiting to be set for execution is being processed, jobs in the queue are sorted by priority and queueing time. Once SLURM system indicates that either a new job has appeared or one of the jobs finished its execution and deallocated resources, the algorithm deletes the job from the queue and sets it for execution; it is

only possible if enough nodes to start that job is available. Otherwise, if there is not enough nodes to start the first job in a queue, the scheduler tries to “pack” a queue, namely it takes the next job from the queue and sets it for execution given that there are enough available nodes and the rights of other jobs in the queue are not being abused (their start will not be delayed) [9, 11]. Two examples on Figure 2 show differences between First Come First Served and Backfill.

2.2 Priority system

As already mentioned job position in queue depends not only on queue time, but also on its priority. Calculation of jobs priorities in SLURM architecture on Lomonosov supercomputer is within the competence of Multifactor Priority plugin, a special plugin from standard configuration of a resource manager. The plugin carries out a system of dynamic priorities. A job’s priority depends on five factors, each of them has weight provided by system administrator during the process of system setup. This five factors in the Multi-factor Job Priority plugin that influence job priority are: age (the length of time a job has been waiting in the queue, eligible to be scheduled), fair-share (the difference between the portion of the computing resource that has been promised and the amount of resources that has been consumed), job size (the number of nodes or CPUs a job is allocated), partition (a factor associated with each node partition), QOS (a factor associated with each Quality Of Service).

Value of each particular job priority is a sum of 5 terms: (PriorityWeightAge) * (age_factor), (PriorityWeightFairshare) * (fairshare_factor), (PriorityWeightJobSize) * (job_size_factor), (PriorityWeightPartition) * (partition_factor), (PriorityWeightQOS) * (QOS_factor).

Table 2. Lomonosov supercomputer Multifactor Priority plugin configuration.

PriorityType=priority/multifactor
PriorityDecayHalfLife=14-0
PriorityWeightFairshare=0
PriorityWeightAge=10000 # default 1000
PriorityWeightPartition=10000000 # default 10000
PriorityWeightJobSize=0 # default 1000
PriorityMaxAge=3-0

If a system administrator is dissatisfied either with standard scheduling algorithms or priority system, his own plugin could be easily put into service using basic interfaces (wiki and wiki2), which are provided by SLURM.

2.3 Lomonosov supercomputer usage experience

In spite of all these advantages of SLURM, RCC system administrators met some troubles controlling Lomonosov supercomputer with its 7200 nodes. There are about 20 users working in any given moment of time and about 1000 users working in a year. Typically there are about 100 running jobs and 200 queued jobs at once. Here we met several troubles and limitations of SLURM.

One of the serious troubles is that each user waits too long time to start his job even using backfill scheduler. It causes very common situation: each user queues several jobs into queue and waits. When user's first job is started, the top of queue is filled by his other jobs, so most of the users wait for one. This trouble cannot be solved by setting limit on maximum running jobs – efficiency probably would be decreased significantly, when the queue contains of many short or/and small jobs.

We propose such new type of limit as CPU-hours per user. (CPU-hours is calculated as time that t CPUs has been allocated to specific tasks of one user multiplied by the number of CPUs allocated.) It allows to manage amount of CPU-hours (or core-hours, or node-hours) of all running and in-queue

jobs for each user. So, every user can run e.g. one job for all nodes of partition for short time, or several small jobs which use a half of all nodes.

Another SLURM disadvantage is inability to set the most limits for partitions by default. E.g. maximum number of running jobs can be set only for the whole cluster on per-user. If we have about 1000 users, limit value (called *account* in SLURM) for each user is added for approximately 1 minute. During this operation no client command (sbatch, sinfo, etc.) can be performed.

This means, that built-in SLURM limits managing is completely unusable for our cases and we need to reimplement it.

To improve efficiency of Lomonosov supercomputer use we need to add some other features to the scheduler. In order to advance the transparency of a priority system it was concluded to abandon the Multifactor plugin and add a new priority system to the external scheduler. The new system is based on user layering — administrator arranges priority of users (from 0 to 65536 in case of Lomonosov supercomputer), forming groups of high- and low-priority users. Annual cluster users survey has presently proved to be a very effective tool using which helps to put unscrupulous users down in priority and raise the valuable ones.

The following updates were introduced to optimize operation of SLURM scheduler:

1. Tracking and control of CPU-hours requested by every user;
2. Transparent priority system with ability to set up in the real-time mode;
3. Ability to use nodes from different partitions for users with certain level of priority;
4. Adding time quotas: fixed number of processor hours per week/month/year;
5. Ability to use different scheduling algorithms in separate queues.

All of these points cannot be implemented without extending SLURM's functionality. SLURM supports APIs for 3rd-party and external schedulers. There are several commercial schedulers, like MOAB [13], which can be used with SLURM. They have very wide abilities, but they also have very high cost. So we didn't use them. Another supported external scheduler is MAUI [14], but it is not developed and will be closed in near future by owner company (which is developing MOAB now). By this reason using MAUI is not safe, and in addition it does not provide all necessary features.

As for opensource of freely available schedulers for SLURM, we found only two projects, which try to implement SLURM plugins. First project is named «slurm-spank-plugins» [15] and it is not developed since 2011. It contains 5 plugins for different additional features, but not for scheduler.

Second project is named «IPSCHED» [16], it is aimed for support of new SLURM scheduler features, mainly heterogeneous CPU-GPU support, and don't have abilities we need.

We did not found any other 3rd-party schedulers for SLURM or publications about them. In summary, it was decided to implement our own scheduler to use Lomonosov supercomputer with its high workload and growing user base most effectively.

3. External SLURM scheduler

Now and then SLURM development team is producing a new version of manager and thus our features must be portable. All our new updates which are introduced in the SLURM code ought to be transportable to a new version of manager in a reasonable time. This main requirement leads to a lot of side limitations with which we had to deal. For instance, changes cannot be made in the system core (background programs slurmd and slurmctld), as this would extremely slow down the updating process; hence, modification of external modules is much more efficient.

The second major challenge is inability to use built-in methods of storing new added updates' configuration. Therefore, it is necessary to write a different data store which would be used for a correct operation of new updates.

In this case, the most optimal solution for the realization of all the additional features is to develop an external scheduler, which will use a basic interface wiki2.

The wiki2 interface was initially developed to support an external scheduler Moab® HPC Suite. On the outside it is similar to the internal scheduler plugin and uses the same primitives and functions, however, the difference is that the scheduling logic, which is fully within the competence of an external module, is missing.

The operation algorithm of the wiki2 is based on a message-passing with an external scheduler via sockets. There are two communication channels between the wiki2 and the external scheduler: the first one serves to transmit messages and receive answers, the second one informs the external scheduler about events in the system [8].

Basic instruction set of an external scheduler includes the following: CANCELJOB, GETJOBS, JOBMODIFY, NOTIFYJOB, STARTJOB, INITIALIZE, REQUEUEJOB, SUSPENDJOB, RESUMEJOB, SIGNALJOB. Each of them requests some kind of information from the wiki2 or passes directions to change state of a particular job. In the channel of "events" the external scheduler can receive only two types of messages: "1234" or "1235". First type corresponds to change of job option; second type refers to the change in structure/configuration of SLURM [12].

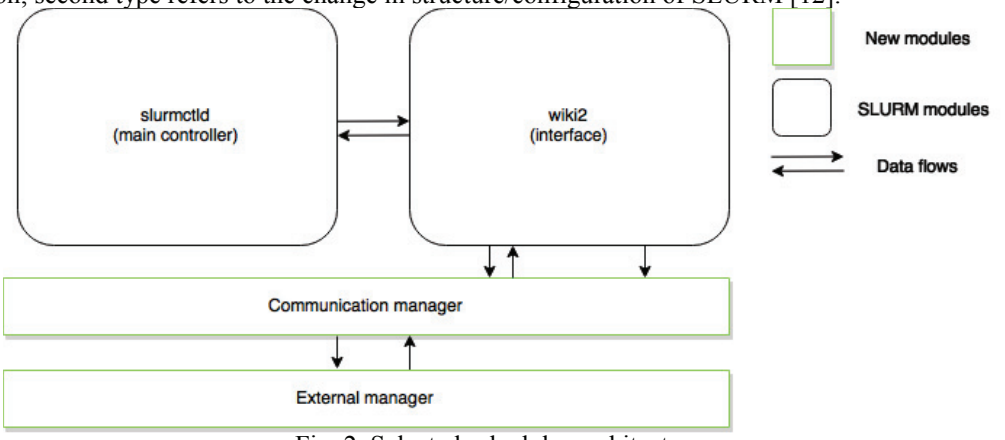

Fig. 2. Selected scheduler architecture.

Formal problem statement for realization of the external scheduler is based on a data set, which is provided by the wiki2 interface.

Inputs:

- A set of queued jobs and their properties (number, time of queueing, user, who put the job in a queue, required number of nodes, limits, etc.);
- State of cluster at this very moment (a set of cluster nodes, number of free nodes to start jobs, etc.).

Outputs:

- Sequence of jobs to be started on a cluster;
- Starting corresponding jobs.

The external scheduler which uses special wiki2 interface (as shown on Figure 2) to communicate with SLURM has been developed. It uses standard wiki2 features and messages (e.g. CANCELJOB, GETJOBS, JOBMODIFY, etc.) Scheduling cycle of external manager is shown on Table 3. Module is written in C++ programming language involving libconfig library and SLURM API functions. The backfill algorithm and aforesaid new features are implemented.

Table 3. External manager scheduling cycle.

1.	Receive data from SLURM;
2.	Remove jobs from queue that exceeded personal users CPU-time limit.

3.	Sort jobs by priority;
4.	Set jobs for execution with respect to their statuses;
5.	Once the next job cannot be set for execution, start packing phase;
6.	When the packing phase is finished, go to standby mode. (After predefined timeout or after receiving event message cycle starts again.)

Packing phase key idea : lift small jobs in a queue but without compromising big ones. More details of this algorithm part is shown on the Table 4.

Table 4. Packing phase.

1.	Calculate time of setting for execution a job which is lacking computational nodes;
2.	Try to launch jobs that have enough space for operating on cluster in a priority order. In the meantime key idea should be taken into account: if the time requested by packing program is larger than the time until a big job would be set, quit the packing algorithm.

4. Performance of the proposed solution

Proposed solution was tested on historical data array of queue “regular4”. (The biggest queue of Lomonosov supercomputer.) CPU hours limit was chosen to allow users to comfortably use the power of the system, but at the same time restrict the ability of almost exclusively occupy part of the cluster. For example, for queue “regular4” time limit value equals to 81920 CPU-hours. This limit gave us ability to reallocate up to 2.7% of the monthly CPU time on jobs of other users, which made it possible to increase the number of users resourced.

Also, external module with a new scheduler plugin has been tested compared to standard SLURM backfill plugin on historical data queue “regular4” Lomonosov supercomputer, and showed good acceleration for first job start time of each user (up to ~9% compared to standard SLURM backfill plugin), while the total start time of jobs almost has been decreased just for a negligibly small amount of time (only 0.2% compared to standard SLURM backfill plugin). Comprehension of SLURM with standard Backfill and SLURM with our external scheduler is shown on Table 5. For testing we used SLURM option, which allows to emulate a large cluster. The same set of jobs were running on two sets of settings: standard SLURM backfill plugin and proposed backfill with CPU-hours quotas. Sets of problems is a real set of jobs, which were launched between October (2014) and December (2014) in queue “regular4” on Lomonosov supercomputer. Almost without falling the overall performance of the scheduling we have achieved visible results in speeding up the start time of the first tasks of each user.

Table 5. Tests on real datasets.

Experiments	Dataset 1 (Acceleration)	
	Delay (user)	Delay (job)
Backfill	0,998	1
Backfill + CPU-hours limit	1	0,907

5. Conclusion and Future Work

Proposed solution has been developed to solve several troubles we met using SLURM. It can be used by any user groups which use SLURM to manage supercomputer. The solution is open source and can be modified for any requirements. It has high portability because of using wiki2 protocol and does not use any SLURM features, which can be changed from version to version.

By using our scheduler efficiency of supercomputer use comparing to standard SLURM schedulers can be raised; new introduced abilities make supercomputer control more flexible.

Now the work is in progress and new features are being developed. New flexible priorities mechanism and per-period CPU-hours quotation are the first priority to develop.

Acknowledgment

This material is based upon the work supported by the Ministry of Education and Science of the Russian Federation (Agreement N14.607.21.0006, unique identifier RFMEFI60714X0006).

References

[1] M. Jette, M. G. (2003). SLURM: Simple Linux Utility for Resource Management. Proceedings of ClusterWorld Conference and Expo. San-Jose, California.

[2] Slurm Workload Manager (2015). Retrieved from http://slurm.schedmd.com/slurm.html

[3] V. Sadovnichy, A. T. (2013). “Lomonosov”: Supercomputing at Moscow State University. In Contemporary High Performance Computing: From Petascale toward Exascale (pp. 283-307).

[4] Lomonosov — T-Platforms (2015) Retrieved from http://www.top500.org/system/177421.

[5] Top50 rating (2015). Retrieved from http://top50.supercomputers.ru/?page=rating

[6] M. Jones (2012). Optimization of resource management using supercomputers SLURM. Retrieved from http://www.ibm.com/developerworks/ru/library/l-slurm-utility/

[7] TOP500 rating (2015). Retrieved from http://www.top500.org/lists/2014/11/

[8] D. Jackson, Q. S., M. C. (2007). Core Algorithms of the Maui Scheduler. Brigham Young University, Provo, Utah.

[9] Q. Snell, M. C., D. J., C. G. (2000). The performance impact of advance reservation metascheduling. Lecture Notes in Computer Science:Job Scheduling Strategiew for Parallel Processing, 1911.

[10] D. Lipari (2012). The SLURM Scheduler Design. Retrieved from http://slurm.schedmd.com/slurm_ug_2012/SUG-2012-Scheduling.pdf

[11] R. Baraglia, G. C., M. P., D. P., L. R., A. D. T. (2008). Backfilling Strategies for Scheduling Streams of Jobs On Computational Farms. Making Grids Work, II. (pp. 103-115).

[12] M. Novotny (2009). Job scheduling with the SLURM resource manager. Retrieved from https://is.muni.cz/th/173052/fi_b_b1/thesis.pdf

[13] Moab HPC SUIT (2015). Retrieved from http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-suite-enterprise-edition/

[14] Maui cluster scheduler (2015). Retrieved from http://www.adaptivecomputing.com/support/download-center/maui-cluster-scheduler/

[15] Slurm-spank-plugins (2015). Retrieved from https://code.google.com/p/slurm-spank-plugins/w/list

[16] Slurm-ipsched (2015). Retrieved from http://code.google.com/p/slurm-ipsched/