Job Scheduling in Homogeneous Distributed Systems

P. OSIAL
posial@wat.edu.pl

Institute of Computer and Information Systems
Faculty of Cybernetics, Military University of Technology
Kaliskiego Str. 2, 00-908 Warsaw, Poland

Today's world demands a lot of computing power for many different applications. Distributed systems offer this with their advantages. High-performance computing clusters are suitable for running different kinds of jobs like tightly coupled parallel and distributed applications. The queuing system is used to organize tasks and allocate adequate resources at appropriate time intervals. One of the fundamental elements in scheduling tasks is to determine the type and characteristics of tasks that will run in a distributed system. The scheduling algorithm is responsible for the proper assignment of these tasks, to the available resources of a particular node. The most important advantage of using the job scheduler in a homogeneous environment is the fact that the scheduler can omit checking of various parameters. Job scheduling aspects in homogeneous HPC clusters environments is presented in this paper. The grid engine was used as a case study for testing common used algorithms for job scheduling. This example showed the problems that may occur when scheduling tasks, depending on the type and quantity of tasks running. The basic algorithm used in this case does not generally meet their function. Complicated cases require more complex algorithms, taking into consideration proper resources utilization.

Keywords: scheduling, homogeneous, distributed

1. Introduction

Distributed systems have a wide range of applications. Today's world demands a lot of computing power for many different applications. Distributed systems offer this with their advantages like:

- high scalability
- high availability
- very good performance
- transparency
- high reliability.

Professional distributed systems consist of clusters built from dedicated multiprocessors servers (nodes) interconnected with a fast network. There are two main groups of clusters [1]:

- High-throughput computing clusters − connecting a wide group of nodes connected by low-end interconnects, which role is to maintain redundancy and high-availability of resources.
- High-performance computing clusters − high-end nodes have large computing power, which is connected with very fast, low latency and wide bandwidth interconnection. The role of this class of systems is to provide maximum performance for demand in terms of computing power and interprocess communication applications.

Scheduling requirements for each of these groups is different. Computing clusters have one common goal, which is the enlargement of the system throughput. This is understood as the number of completed tasks in a given unit of time. In high-throughput computing clusters this is the main objective of the system implemented by the load distribution between the different nodes. Best suitable job types for this class of systems are loosely coupled distributed and parallel programs.

High-performance computing clusters are suitable for tightly coupled parallel and distributed applications. These types of applications have special requirements for the communication and synchronization. Therefore, this class of clusters requires additional conditions that must be taken into consideration. These additional performance objectives are to minimize the job execution time, reduction of communication and other overheads [2]. On the other hand, maximization of resource usage.

Additionally to these two classes, there are two types of clusters in terms of system architecture: homogeneous and heterogeneous system. If the environment is completely uniform, in terms of hardware and software, then we will talk about the homogenous environment, otherwise the heterogeneous environment.

The queuing system is used to organize tasks and allocate adequate resources at appropriate time intervals. It also consists a minimum of three elements together, interacting with each other: job scheduler, resource manager and accounting manager. As shown in figure 1, the scheduler is responsible for creating a queue of tasks and mapping it to individual free system resources.

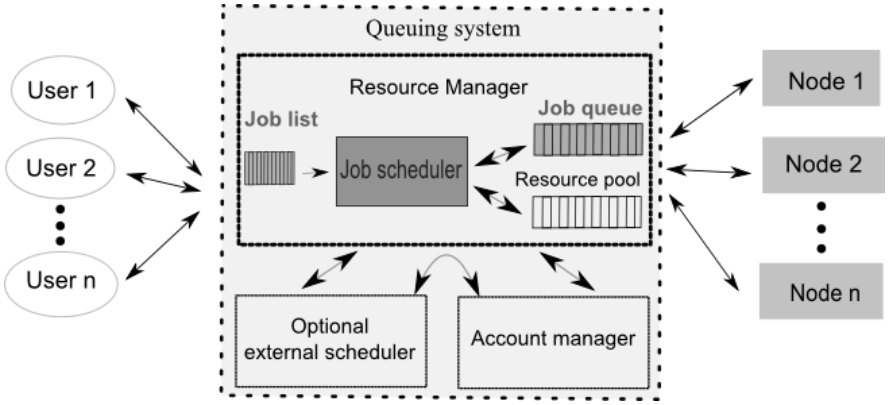

Figure 1. Cluster queuing system

The second element is the resource manager, passing information on current time system resources utilization. The next module is an account manager keeping information on users like: privileges, job priorities, restrictions and limits. System administrators can setup separate privileges for any user. Together they form a complete system interacting with each other. Users can check for resource availability in a cluster, submit a job and check the progress of the task. When a user submits a new job to the queuing system, then the job scheduler checks the status of resources on individual computers through the resource manager. Taking into consideration information on priority tasks and limitations set by the system administrator, schedules of the job assigning resources in a queue are used during job execution. Most of the queuing system allows using external schedulers and makes it possible to fit the individual needs of users.

2. Task Characteristics

One of the fundamental elements in scheduling tasks is to determine the type of tasks that will run in a distributed system. Depending on the type of tasks, which are submitted, there should be different rules for allocating resources for these tasks. The list below presents the basic types of tasks:

- Single − job running on any machine in a cluster, which have a compatible processor architecture
- Serial − set of tasks associated with each other, order of execution is essential because of the necessity to use intermediate results of the previous job to continue the calculation. It will be run on one or more machines at the same time, if the order condition is satisfied
- Symmetric Multiprocessing (SMP) − task composed of several subtasks running simultaneously within a single machine. Uses the n available of nc cores in one machine $n=\{2,\ldots,nc\}$
- Distributed tasks − Distributed task execution using multiple threads that communicate among themselves. Such a task can be completed on n cores with the ac all available $n=\{2,...,ac\}$.

Each of the above task groups may have different needs in terms of hardware resources, such as necessary system memory to run appropriate or local temporary space for intermediate results. These types of requirements are individual for each program and type of data, which operates in a particular case. The most demanding, in terms of scheduling, are distributed tasks, where interprocess communication plays an important role in the overall application performance. If the path between the most distant processors is long, then the performance of applications will be worse.

Figure 2 shows the physical connection between machines.

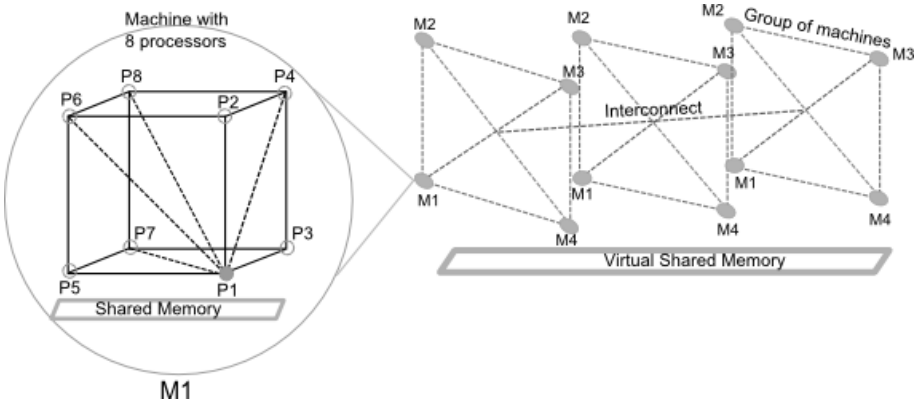

Figure 2. Interprocess Communication

Communication between the processors within the machine is negligible, so these delays are not taken into consideration. However, after leaving the machine outside, the communication delays are strictly dependent on the type and quantity of connections between intermediary devices. For this reason, the subprocess tasks should be distributed from each other with the least distance that is within a machine or a common switch.

Another set of restrictions for each type of jobs are based on software requirements. Each task has different requirements. However, they possess a common set of requirements by the type of task. They can be divided as follows:

- single and serial jobs − generally do not require additional services to work properly
- SMP jobs − depending on the method of compilation in most cases require additional system libraries, and monitoring services working on the local machine
- distributed tasks − requires for proper operation of additional services, both for starting their associated libraries, and unit of services for monitoring of jobs that reside in each machine on which task or part of it is executed.

3. Job Scheduling

In the scheduling system, we can determine three main components [3]: waiting queue, the current schedule and scheduling algorithm. The current schedule provides information regarding the assignment of system resources like the number of processor cores and possibilities of their allocation to each task. The tasks which had been added to the system for execution, but did not start yet, are in the waiting queue. The scheduling algorithm is responsible for the assignment of tasks to the resources of a particular node.

The user adding the job to the queue has to wait to execute its task until other jobs release system resources. The time spent in the queue (wait time) depends on many factors for example: the priority of tasks, the amount of free resources and number of queued tasks.

Most important advantages of using the job scheduler in homogeneous environments is the fact that the scheduler can omit the checking of various parameters. These values are known from the start of the scheduling process:

1. Resource specification for every node:
   - CPU
   - Memory
   - Mass memory
   - Interconnect type.
2. Software specification:
   - Operating system
   - Installed software
   - Configuration.

These elements are constant, and there is no need to check their status at the time of scheduling. The elements that must be taken into consideration are: the data on the current memory consumption and available processors. Knowledge of the hardware architecture allows to get rid of additional conditions.

3.1. Algorithms

Scientific society sacrificed much time and effort to explore the aspects of scheduling tasks in parallel and distributed systems. As a result of these efforts, several algorithms were created. Some of them have been implemented in commercial and open source schedulers.

These algorithms can be divided into two main groups: space-sharing and time-sharing.

Time-sharing algorithms divide the available CPU time for discrete time slots assigned to the individual tasks. Thanks to this solution it is possible to share multiple tasks simultaneously on the same hardware. Algorithm space-sharing on the other hand assures the allocation of required resources for particular tasks throughout the entire execution time. In most cases, the cluster job scheduler uses space-sharing algorithms.

The basic, simple, commonly used, space--sharing algorithms are: First-Come First-Serve (FCFS), First In First Out (FIFO), Round-Robin (RR), Shortest Job First (SJF), Longest Job First (LJF).

FCFS and FIFO as the names suggest performe tasks in order, then they are added to the queue. Round-Robin − assigns tasks to individual machines after adding to queues in a cyclic and equal manner. Those are simple algorithms, which efficiency is acceptable for low loaded systems.

The SJF algorithm uses the declared time of task completion to prioritize jobs in the queue. It sorts them from the shortest to longest and additionally, it is able to gain a good turnaround time for small tasks. This strategy causes the delay of long executing jobs.

On the other hand, the LJF at the first run of jobs with the longest execution time. Maximizing usage of resources, increases the turnaround time of short jobs.

To complement these basic algorithms, most commonly used schedulers use additional techniques, such as advance reservations, backfill and fair-share.

Advance reservation techniques use information provided by the user about the predicted executing time of the job. After that it reserves the required resources on selected machines for a specific job and generates a schedule.

Backfilling techniques improve space--sharing algorithms by filing small low priority jobs into scheduling gaps. This algorithm does no changes of the previous schedule, but it complements the gaps that arise for scheduling tasks with high priority jobs. The requirement for usage of this algorithm is information about added to the queue tasks, such as priority and estimated time of execution. Information on the execution time must be provided by the user running the task, otherwise the usage of this algorithm is not possible. Figure 3 shows a sample of a set of tasks using the following scheduling algorithms.

The fair-share algorithm uses collected historical data regarding the job execution. On this basis the priority of tasks dynamically changes when the system is heavily loaded, in order to maintain a fair resource allocation between users.

These algorithms are efficient and simple to implement. They are widely used in real applications. In a small group of applications there are many unimplemented or used algorithms.

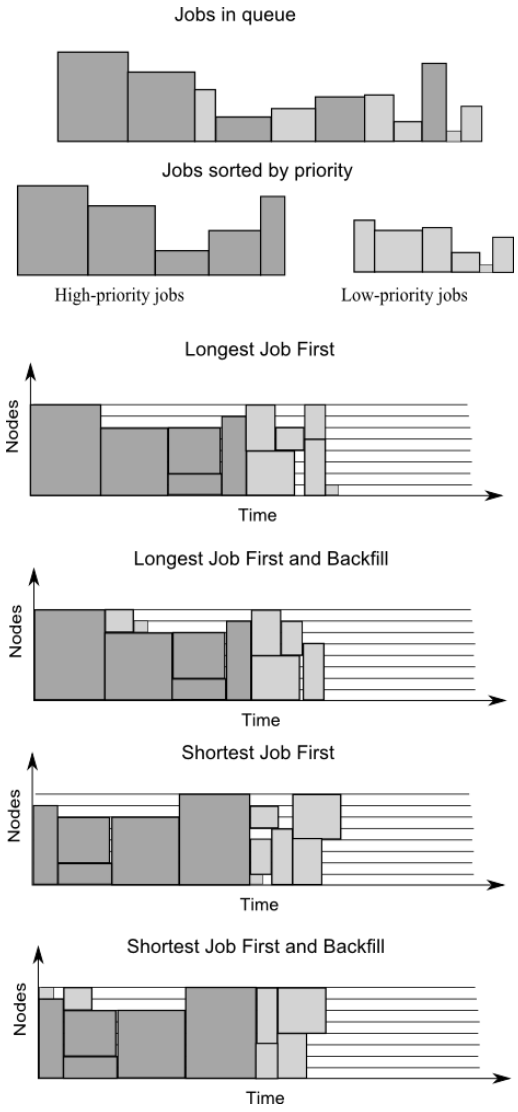

Figure 3. Job scheduling algorithms use example

Generally, they are fairly complex, require more resources and time to arrange the schedule. In a real application, the superiority of these algorithms over the combination of the algorithms above has not been proved.

3.2. Scheduler Attributes

Current schedulers must meet several features in order to be used in real applications. The main attributes that each scheduling system must be able to provide are:

1. Support for many job processing algorithms such as: FCFS, FIFO, LJF, SJF, advanced reservations, backfilling and fair-share.
2. Ability to dynamic switching between algorithms depending on needs (type of job, group members).
3. Capability to integrate with popular tasks managers such as SGE, OpenPBS, LSF.
4. Taking into account different nodes of the hardware architecture (CPU) and interconnection capabilities between nodes (bandwidth), in order to best utilitize available resources.
5. Performance to minimize the overhead of management job processes. Complex scheduling algorithms need more time. Task scheduling must be minimized for system efficiency.
6. Dynamic change in information on available resources. To add and remove machines from the pool without having to stop or restart the system.
7. Scalability of the system allows them to operate on small to very large systems, and seamless support for thousands of simultaneous tasks.
8. Support of different types of tasks such as: single, parallel, batch, distributed, serial, interactive and not interactive with comparable performance.
9. Check pointing support allows suspending tasks, to save their state, and resume on the same or another machine. This makes it possible to preserve and restore intermediate results, generated by a job in case of hardware or software failure.
10. Resistance to incorrect functioning of client applications or hardware errors of subsequent nodes. Single machine problems do not cause faulty operation of the entire system. Flexible software updates without having to shut down the entire system or suspending user tasks.

4. Case Study: Grid Engine

Grid Engine (GE) is an open source advanced job manager system based on the commercial Sun Grid Engine (SGE) as part of the Solaris Enterprise System. It has a well-developed console and clear graphical interface and monitor currently loaded on nodes. Services monitor system resources such as occupancy of processors, memory, disk space, and parameters reflecting the use of these resources by individual users.

Grid Engine supports different types of tasks including distributed jobs, thanks to the possibility of tight integration with the most commonly used parallel environments such as LAM or MPICH2. In GE terminology the queue is defined as group of resources, which is divided in to slots (CPU cores). Multiple queues can be assigned to the same hardware resources.

Grid Engine supports many important functions crucial for job management. The most important of these features are:

- advance reservation
- check pointing
- rule-based resource quota control
- distributed resource management application API (DRMAA)
- support for interactive jobs
- online resources
- job Submission Verifier (JSV)
- job and scheduler fault tolerance
- topology-aware scheduling and thread binding
- policy-based resource allocation
- use of multiple scheduling algorithms.

The job after submitting for execution is sent to the scheduler where requirements and priorities are determined. Afterwards the master daemon receives information about adding another job to the execution queue. All information regarding the system, tasks and current state are stored on the server nodes. When the user adds new jobs to the pool, then they have the possibility to specify parameters. Informing how to perform jobs and requirements to be fulfilled by the system in order for the task to be executed properly. Fundamental parameters that should be given when a job is submitted are:

- type of Job − specified by executing a job in an appropriate queue
- number of CPU (slots) needed for execution if the job is SMP or distributed.
- estimated job completion time.

Selected optional parameters that may be passed to the job manager:

- specific job name
- Software requirements (available software licenses etc.)
- hardware requirements (amount of memory and storage resources)
- localization of data results (default local directory)

- specification of a particular executive node.

More parameters will be given, the more effective management system will be in job scheduling and execution. GE takes into consideration available parameters and on this basis attempts to arrange the schedule. Information about resources and job execution are acquired on execution nodes and sent to server nodes.

One of the biggest GE advantages is the fact that cluster administrators can add new functionality to the system [4]. Thanks to that the system is based on scripts, it is possible for the programming of new functionality such as load sensors to the particular queue or whole system. As soon as the new queue is defined and the resources are assigned to it are free. The system automatically checks for jobs that may run in this queue, and running a job with the highest priority or longest waiting time. Jobs are started on nodes by executive daemons running on every executive node. These services are monitoring progress, reporting system load memory consumption and other configurable parameters, to the master daemon on the server node.

Grid engine supports check pointing and migration. Supported will be the kernel-level and user-level check pointing, if the application supports it or by using third-party software like Berkeley Lab Checkpoint/Restart (BLCR). Another useful feature is the calendar so it can be planned when the queue will be available to users for job submissions or when it will be inaccessible. For example, for the maintenance of equipment, software update or adding another resource pool for a short period of time when system is heavily loaded etc. In the users point of view GE provides an intuitive user-friendly graphical environment to manage tasks. Obviously the console command set is also available who provide a larger set of functionality for the advanced users.

The Grid Engine has some disadvantages the biggest of them is the fact that the scheduling module has significant limitations in the definition of scheduling policies. Mechanisms for preemptive scheduling and making reservations are complicated and need knowledge about the principle of scheduling the mechanism functions. Supporting parallel tasks in GE is more complicated than in other schedulers. Nevertheless, tight integration is fully supported after the correct manual configuration of the entire distributed environment starts scripts. Grid Engine software is constantly developed and improved.

One of the most important elements affecting the scheduling is the type and characteristics of the tasks that are executed. The easiest cases are the uniform type of tasks, when the same type of tasks are run with similar characteristics. The most difficult case is one where in a single system multiple tasks of different types are running with varied characteristics. Despite these limitations this system is fully working and efficient for most applications. The system fulfills its function the best for applications with common types of tasks. For example, system scheduling of single or serial tasks like the one shown on figure 4. But in more complex cases, when the environment does not have the resources dedicated to each type of task the situation is a bit different. For proper scheduling of tasks it is necessary to use an advanced reservation mechanism and the changing of priorities for individual tasks and task types. The result is often not in accordance with the awaited result.

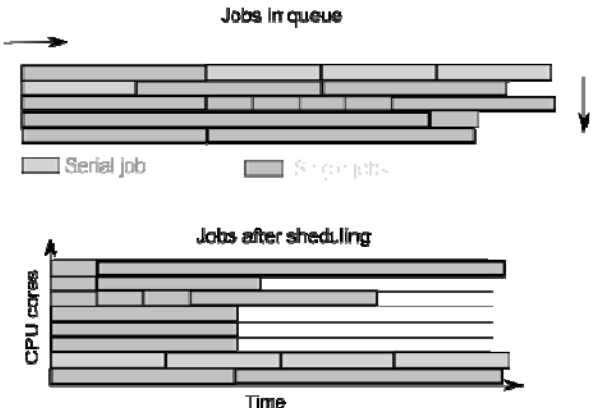

Figure 4. Scheduling single and serial jobs using default queue settings in GE

Each environment has different characteristic. The percentage of job types running in a cluster system depends on the type of tasks that are run by users and the level of sophistication, which software offer. The same software might contribute to the load of 100% or less than 50%. The time of task execution also varies depending on the characteristics of the input data. The greatest demanding and also difficult to predict case are applications used for scientific calculations. Parallel applications, depending on how their source code was written, have a different degree of parallelization. For most cases, real measurable acceleration of the application is visible with the use of up to the eight sub-processes. There are of course exceptions to this rule, currently well-written applications are scalable to 16 or even 64 and

more sub-processes, each running on an individual CPU. Everything depends on the degree of parallelization, which the running application has. Compilation efficiency and hardware parameters such as interconnect latency etc. on execution nodes. For this reason, parallel tasks should be executed on the smallest possible number of accessible nodes. The optimal solution is to use one machine as far as the amount of available resources allow it.

For example the encountered in a production environment is a set of jobs. That jobs run by a group of users with the same priority. The relationship between the types of tasks is as follows: single/serial jobs 50%, SMP jobs 10% and parallel tasks 40%. Most encounter problems in job scheduling with GE default algorithms without advanced reservation for the particular tasks are shown on figure 5. Figure 5a presents a case, in which we are dealing with combinations of different tasks without setting an execution deadline on parallel tasks. Large distributed jobs in this case are subject to starvation during heavy system load and the continuous inflow of single tasks. Figure 5b shows the same scenario with the preset deadline on parallel tasks. In this situation the delay of a distributed task is very large and depends on the deadline time. Figure 5c, 5d shows a typical problem when running large distributed jobs where subtasks distribution on the resources is not optimal. One of the poorer cases is 5c where two parallel jobs and a single task are running on the same node. The worst case scenario is shown on figure 5d. Parallel tasks are running simultaneously, on all nodes with single and serial jobs. Performance of parallel jobs depends on many factors. One of the most significant factor, especially for large amounts of data procession is interprocess communication. The more demanding task will run on one node the bigger will be the cost of communication. By analogy the more hosts are involved in this process the higher the cost. The second factor is the overhead caused by the intensive use of system resources. Thereby decreasing the power by redirecting resources to operate the I/O. Depending on the application gap the performances may vary from 4% to 53%.

Many factors influence the estimating execution time. Because of that using advanced reservation to determine the correct scheduling of tasks is not the optimal solution. Often the resources are reserved, but the job ends running before the time of its execution or after the expected time. This causes gaps and shifts in the schedule.

5. Conclusions and Further Research

Distributed systems are very well known in today's world of science. They where researched for a couple of years and most aspects of this problematic field was well described. According to the fact, that some aspects are still not well known, the main concept of this system consists many of loosely interconnected processors. Distributed shared memory is used to store and share data.

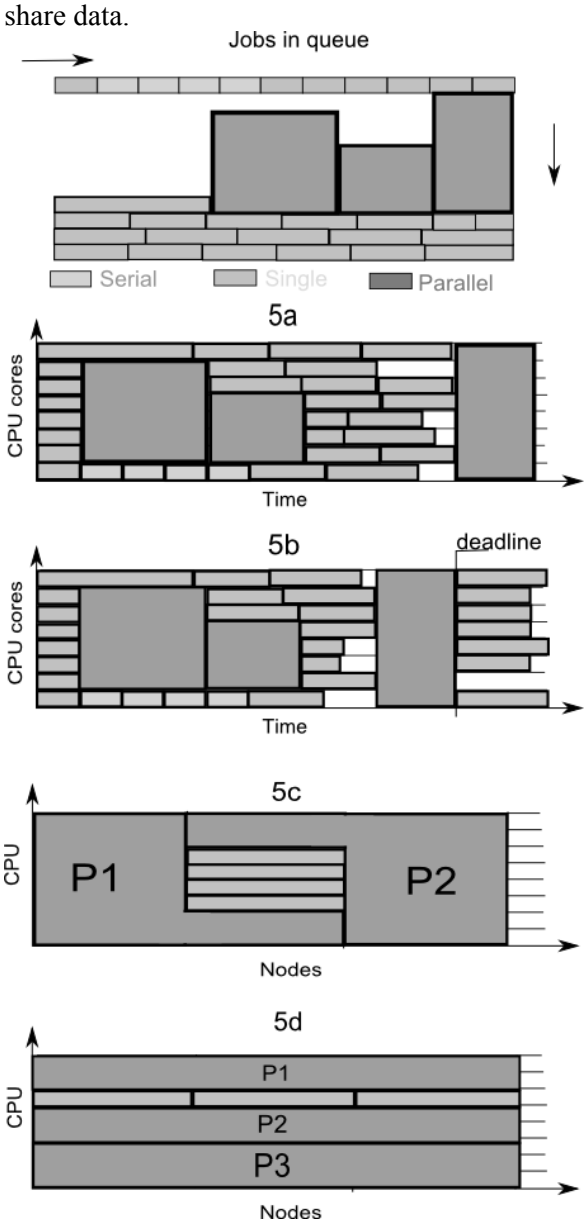

Figure 5. Scheduling issues

Optimization of such systems is reduced to optimal scheduling, of running jobs on this system.

They are many goals in optimization, two most common used are. minimization of

completion times of jobs and maximization usage of available resources. In most cases both criteria are used simultaneously. Proper scheduling will depend on the characteristics of the tasks performed, and the type of distributed environment. In the considered environment, high performance computing, parallel tasks are an important part of running a job. These tasks require special attention due to the number of requirements to be met by the system in order for such a task to finish correctly. Moreover, the time that the job is executed should be reasonably short, in accordance with expectations. Systems, which are operating only single and serial jobs do not require advanced scheduling techniques. Using a typical SJF algorithm with backfilling is in most cases close to an optimal solution acceptable in real applications. Scheduling more complex cases, where there are additional task types, parallel and SMP requires the use of more complex techniques for the proper distribution of jobs. Fulfilling inflicted conditions and criteria for optimization.

A basic algorithm used in this case does not generally meet their function even after completion of their functionality by advanced reservation and fare share. The administrator intervention is indispensable to enable proper operation of scheduler. This intervention is modification of manually adjustable parameters such as job weight and deadline time. Such applications require more complex algorithms, taking into consideration proper resources utilization. For homogeneous systems, this situation is so much better that some of the parameters are fixed, which partly reduces the complexity, by removing part of the restrictions, related to the variety of resources. The task scheduling problem is NP-hard for two nodes. So, for large systems with tens of nodes, this problem is possible to optimize only by approximations. However, it must be brought an appropriate set of conditions, and an optimization function that enables the creation of a task scheduling algorithm.

The logical extension to this paper is the examination of the best available scheduling algorithms to this specific environment. Determination of the optimal scheduling algorithm for a complex case, with defined constraints make possible to implement in Grid Engine environment.

6. Bibliography

[1] S. Iqbal, R. Gupta, Y. Fang, ”Job Scheduling in HPC Clusters”, *Power Solutions*, 133−136, Dell Inc., February 2005.

[2] H.D. Karatza, R.C. Hilzer, ”Parallel job scheduling in homogeneous distributed systems”, *Simulation*, 79, 5−6, 2003.

[3] C. Franke et al., J. Lepping, U. Schwiegelshohn, ”Greedy scheduling with costume-made objectives”, *Annals of Operations Research*, 180, 145−167, Springer, 2010.

[4] G. Borges et al., ”Sun Grid Engine, a new scheduler for EGEE middleware”, *IBERGRID − Iberian Grid Infrastructure Conference*, 2007.

Harmonogramowanie zadań w homogenicznych systemach rozproszonych

P. OSIAL

W dzisiejszych czasach wzrasta zapotrzebowanie na moc obliczeniową dla szeregu aplikacji. Systemy rozproszone dzięki swym atrybutom są w stanie sprostać tym wymaganiom. Klastry obliczeniowe o wysokiej wydajności tworzą odpowiednie środowisko służące do uruchamiania wielu typów zadań. System kolejkowania umożliwia poprawne rozmieszczanie aplikacji na poszczególnych zasobach, w odpowiednich przedziałach czasowych. Jednym z podstawowych elementów podczas tworzenia harmonogramu jest określenie typu i charakterystyki uruchamianych zadań. Dzięki temu algorytm odpowiedzialny za uszeregowanie zadań jest w stanie poprawnie wykorzystać dostępne zasoby. Jedna z zalet układania harmonogramu zadań w systemie homogenicznym jest możliwość pominięcia sprawdzania szeregu parametrów. W tym artykule badano aspekty szeregowania zadań w jednorodnym środowisku klastrów HPC. Grid Engine został wykorzystany jako studium przypadku do badania najczęściej używanych algorytmów w planowaniu zadań. Przedstawiono problemy mogące występować podczas planowania zadań w zależności od typu oraz ilości zadań. Pokazano również wady podstawowych algorytmów. W przeciwieństwie do zaawansowanych algorytmów, nie spełniały one swej funkcji w skomplikowanych przypadkach.

Słowa kluczowe: harmonogramowanie, homogeniczny, rozproszony.