UDC 255:29.1

IMPLEMENTATION OF SCHEDULING ALGORITHM IN HIGH-PERFORMANCE COMPUTER CLUSTER

J. Skrinarova, M. Krnac
Matej Bel University, Faculty of Natural Sciences, Slovakia
Tajovskeho 40, Banska Bystrica, 97401, Slovakia. E–mail: jarmila.skrinarova@umb.sk
P. Martincova
University of Zilina, Faculty of Management Science and Informatics, Slovakia
Univerzity 8215/1, Zilina, 010 26, Slovakia. E–mail: krtcom@gmail.com, Penka.Martincova @fri.uniza.sk

In our paper we introduce an optimization algorithm for scheduling jobs in high-performance computer cluster. The scheduling approach is based on popular, widely used and efficient Hill Climbing algorithm. We compare two different approaches to scheduling in parallel systems: queue-based and schedule-based scheduling. We implemented suggested algorithm within TORQUE resource manager in real production system.

Key words: HPC scheduling, resource manager, job scheduling, schedule-based scheduling.

ВПРОВАДЖЕННЯ АЛГОРИТМУ ОПЕРАТИВНОГО ПЛАНУВАННЯ ВИСОКОПРОДУКТИВНОГО ОБЧИСЛЮВАЛЬНОГО КЛАСТЕРУ

Дж. Скринарова, М. Крнак
Університет Матея Бела, Факультет природничих наук
ул. Тайовського, 40, Банська Бистриця, 97401, Словаччина. E–mail: jarmila.skrinarova@umb.sk
П. Мартинкова
Жилинський університет, Факультет науки управління та інформатики
Університет 8215/1, Жилина, 010 26, Словаччина. E–mail: krtcom@gmail.com, Penka.Martincova@fri.uniza.sk

Наведено розроблений авторами алгоритм планування завдань високопродуктивного обчислювального кластеру. Методика, яка була взята за основу розробленого алгоритму, базується на ефективному та широко розповсюдженому алгоритмі локального пошуку (Hill Climbing). Авторами проведено порівняльний аналіз двох підходів до планування завдань у паралельних системах: за чергою та за завданням. Запропонований авторами алгоритм було застосовано у менеджері ресурсов TORQUE в умовах реального виробництва.

Ключевые слова: планування високопродуктивних обчислень, планування завдань, планування за завданням.

INTRODUCTION. Parallel and distributed computer architectures in recent time tend to have great field of usage. They are widely used mainly in technical science disciplines, physical, chemical or biological research, astrophysics etc. Simple way how to use a growing potential of multiple computers is to connect them together with network and create a cluster. Clusters can be divided into two groups according to computing hardware and network components:

1. High-throughput cluster – created by large number of computers with low-end network components,
2. High-performance cluster – created by more powerful computers with fast interconnections [1].

PROBLEM STATEMENT OF RESOURCE MANAGEMENT AND SCHEDULING. High performance clusters are primarily suitable for executing parallel programs, which require multiple computing machines or processing units. These computations often need to exchange information during their runtime, which creates the demand to minimize the amount of communication between computing units by effective mapping of tasks to available machines and their processing units. Resource mapping is provided by a scheduler, which is special software running on one of the cluster computers. As shown in figure 1 it communicates with another program, which is responsible for monitoring all the computing nodes, communicates with them and gathers information about their available resources like architecture, number of processing units, physical memory, current workload etc. Scheduler can be either part of resource monitor (internal scheduler), or external program which is able to interact with resource monitor.

Standard structures that are part of resource monitor are:

– Server – acts like an entering point for users to submit their tasks to the system.
– Working node – physical, or virtual machine which runs part of the resource manager responsible for executing tasks on that particular machine.
– Queue – list of jobs ready to be executed.
– Job – executable program with certain input and output. It can consist of multiple tasks which are consequently assigned to processing units of a working node.
– User – represents users of operation system which can have different privileges defined by administrator.

Each job defines its resource requirements. Most common resource requirements are: number of nodes, number of processing units, amount of memory, time required to finish the task. After submitting the job to the system it is queued and waits to be executed. The expectations of users, concerning the time of execution of the job can be expressed by some of these metrics:

– Wait time – time, jobs spend in queued state.
– Turnaround time – time from the submission until the completion of the job.
– Response time – time from the submission until the response to the user.
– Makespan – maximum of total completion times

of sequence of tasks, one of the most interesting parameter in parallel and distributed systems [2].

These requirements often get in contrast with the aims of administrators, who try to effectively use all the resources in the system, thus minimize the total costs of cluster operation. The scheduler which maps the jobs to available machines has to consider many requirements, which makes this process quite difficult.

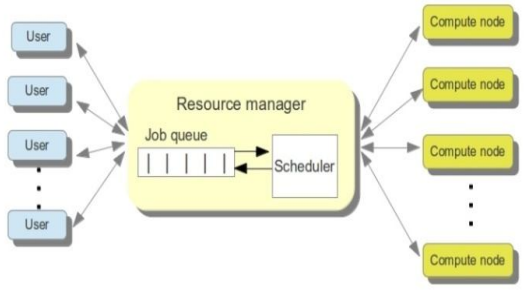

Figure 1 – Resource manager and communication between users and nodes

COMPARING SCHEDULE APROACHES. Generally there exist two different models of mapping jobs to available working nodes: queue-based mapping and schedule-based mapping

Queue-based mapping. Queue-based mapping of jobs to nodes uses queue as basic structure. It is a list of jobs submitted to system by users usually sorted by their submission time. Another sorting strategy can be chosen by system administrator and the most common are: Shorted Job First, Longest Job First, Shortest Execution Time First etc. After submitting the job to a queue it stays there until next scheduling cycle when scheduler selects the job from the queue and tries to determine the best node for running the job. If it succeeds the job is send to be executed on that node and its state is changed to running. Finding the best node is based on certain scheduling policy, configured by administrator. It is a complex process that involves several configurable parameters which can affect the resulting decision. If more queues exist, this algorithm can be modified by selecting and running one job from each queue before trying to assign another job from the same queue. Scheduler with one queue is shown in figure 2.

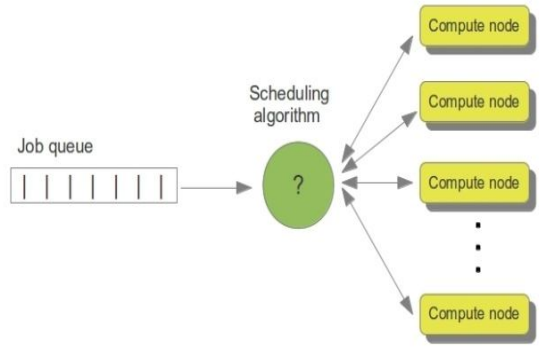

Figure 2 – Queue-based cluster scheduling

Resource manager sends a request to the scheduler in several situations: after new job submission, after job termination or in specified time intervals. This or similar scheduling is used in many production systems such as PBS Pro [3], LSF [4], Sun Grid Engine (SGE) [5], Condor [6], Maui a Moab [7].

Schedule-based mapping.

Opposite to queue-based scheduling system, another approach uses so called schedule, which is basically a list of tasks which are assigned to each computing node, containing information of estimated start and finish time and other characteristics. Creating and sustaining such a schedule is possible in two different ways. First method creates new schedule every time a job is submitted. The schedule can be consequently evaluated and optimized in order to create the most optimum solution for our requirements. Second method creates schedule in the moment of first job submission and every other job is only integrated into this list provided that it stays in the most optimal state after the job becomes part of this schedule.

Both these methods have some advantages and disadvantages. Generating new schedule after each submission can result in increased time consumed by scheduler to find the optimal solution. Especially when dealing with large number of jobs. On the other hand starting from scratch can result in finding even better solution which might not be possible to achieve if keeping the previous schedule. The efficiency of the schedule can be evaluated it by using some of the indicators mentioned above mostly by calculating the makespan. It can be easily calculated based on the schedule and estimated start and end times of the jobs. Schedule-oriented approach to mapping tasks to machines is shown in figure 3.

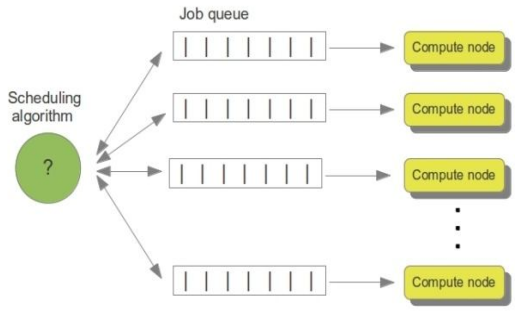

Figure 3 – Schedule-oriented approach

CUSTOM SCHEDULING ALGORITHM IMPLEMENTATION. One of the aims of our research was to implement own scheduling algorithm suitable for the high-performance cluster at our facility. The resource manager used on the cluster called TORQUE [3] includes default scheduler, but can also be extended by compatible external programs. Default scheduling algorithm used in this production system is FIFO with several configurable options.

This algorithm can be efficient for smaller computer clusters which do not require difficult planning and

prediction of system behavior. This mostly concerns the resource utilization based on analyzing the schedule and prediction of future system state. Different schedule can result in differently efficient resource utilization therefore it is desirable to optimize the schedule as much as possible. The number of possible scheduling solutions grows rapidly with the increase of input parameters (mainly number of jobs and number of available nodes). Finding the optimal solution is the problem that can be classified as NP-complete [8] thus its resolving can take unrealistically long time using deterministic algorithm searching over all solution space. However many heuristic algorithms were designed to solve this problem often inspired by nature phenomenon like genetic algorithm, particle swarm optimization, ant colony, or many other: hill climbing, simulated annealing, taboo search etc.

HILL CLIMBING OPTIMIZATION.We present a schedule-based approach to mapping jobs to nodes with Hill Climbing algorithm as an optimization technique. It is a fast-converging method that works with random initial solution which is continually changed to a better solution until it cannot be improved any more. In each step surrounding solutions are examined and the one that improves the current solution the best is chosen and proclaimed the actual best solution. One of the known disadvantages of this algorithm is the chance to get trapped in local extreme of the function. This problem can be solved or at least its effects can be minimized by modifying the algorithm to include random restarts, or some other stochastic behavior. It is shown that it is an effective method for finding local optimal solutions of scheduling problems [9].

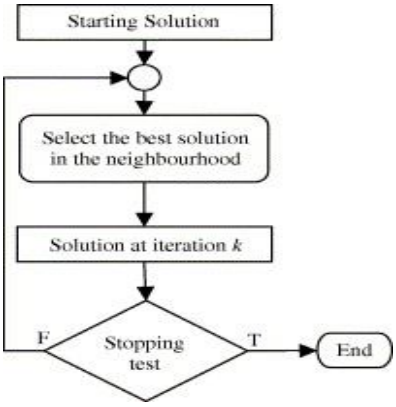

Figure 4 – Simple hill climbing algorithm flowchart

Each job *j* has several attributes:

p_j – processing time. For processing time on machine *i* it is p_{ij}, d_j – due date. Time when job execution stops, even if the job is not finished, w_j – job weight. Express the importance of the job. It can represent the priority of local job before global or the user preferences, s_j – set up time. Time necessary for all the operations required for successful job execution (loading libraries, managing input and output etc.), M_j – machine eligibility restriction. Describes the nodes that can process job *j*. S_j – start time. Time of actual execution start, C_j - completion time. Time when the job execution stops. $pij = C_j - S_j$.

Times d_j and C_j represent real times, p_j and s_j are CPU times. For given schedule we can easily calculate these values for every job J_i. Another indicators that can give us information on how the system behaves are time of completion of job on node C_i, latency $L_j = C_j – d_j$ and positive latency $T_j = max\ \{0,\ C_j – d_j\}$.

The most often used optimum criterion can be determined as$Cmax = MAX\ \{\ \sum\ (C_j – S_j)\ \}$where *Cmax* is the maximum time of completions (makespan). We can easily calculate this value and so we can evaluate the schedule.

CONSIDERING PARALLEL JOBS. In order to be able to effectively create a schedule for parallel jobs we need to design the structures for our scheduler. Graphical representation of schedule that takes parallel jobs into consideration is in figure 5.

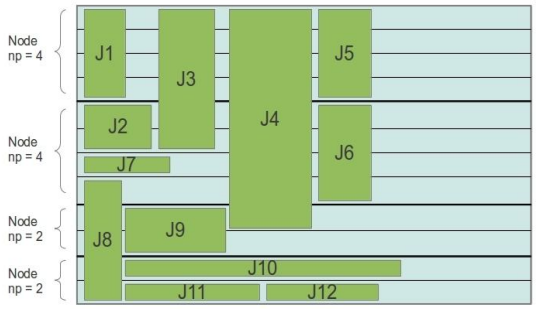

Figure 5 – Example of schedule with parallel jobs

Each node contains one or more processing elements, which can be the same as the actual processors or cores. User can specify their request when submitting job into system either by providing number of required nodes and processing elements, or specifically stating the name of requested node and number of processors. Other requests, which are available to the users, can be: architecture of the node, physical memory, hard disk space, special software required and so on.

The scheduler has to take all these requirements into consideration. Basic structure in our scheduling algorithm is node request which contains information about requested node - name, number of processors, architecture etc.

Each job contains list of node requests which is constructed from user submission requirements. Unless specific hostname of node is provided we assume that job can run on any node that fulfills other criteria. If no value is provided for number of processing units default value of 1 is assumed. Default value for graphical processors is 0.

Proposed algorithm works as follows:

1. Get the list of jobs and construct the node request list for each of them
2. Based on this each jobs node request list assign each node request to specific node. If some information is not provided choose first best suitable node.

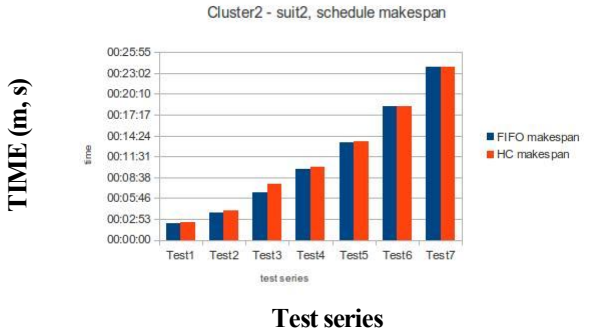

Figure 6 – Schedule makespan

3. Evaluation algorithm:

– For each job j create a list of previous jobs that must finish before job j.

– Count estimated start time for each job by recursively count estimated completion times for each job in its previous job list.

– For each node which has some running jobs count estimated remaining time to finish running jobs.

– Calculate maximal completion time C_i on each node.

– Count maximal completion time C_{max} for current schedule.

4. Try to optimize the schedule using Hill Climbing algorithm. Search all neighboring solutions of initial solutions. Neighboring solution can be obtained by swapping two jobs in original list of jobs.

5. Evaluate schedule and store it if better makespan value was found. Continue from step 2 until no improvement can be found.

ALGORITHM PERFORMANCE. We ran 2 test suites each containing several test series on 2 different clusters to determine the efficiency of proposed algorithm. Technical information about the testing clusters are shown in table 1.

Table 1 – Technical information about testing clusters

Cluster	Cluster1	Cluster2
Architecture of nodes	x86_64	x86_64
OS	Ubuntu Server	Debian 4.0
RAM/ node	512MB	1GB - 4GB
HDD	6GB / node	100GB / node
Nodes/cpus	4/1	5/2, 2/1

Each cluster uses TORQUE 3.0.6 as resource manager and has MPI implementation with TORQUE support installed in order to be able to run parallel programs.

We ran 2 suites of tests on each of these clusters. First suite – suite1- consisted of 6 test series of 10, 20, 30, 40, 50 and 60 jobs of different length and node requirements randomly sorted. In each series all the jobs were submitted to the system before the scheduler was started.

After the scheduler startup the initial schedule was generated and remained valid for the whole test. We recorded time necessary for generating this schedule and measured the time needed to complete all jobs submitted in one test series. We also recorded the average and maximum turnaround time for each job and for each test series.

Second test suite – suite 2 – contains 7 test series with randomly sorted jobs with various time and resource requirements similar to the ones in test suite 1. We continuously kept submitting these jobs in random intervals and again recorder the total completion time, maximum and average turnaround time and other statistics like the time of schedule generation after each job submission. The result of these test are shown in figures 6, 7 and 8.

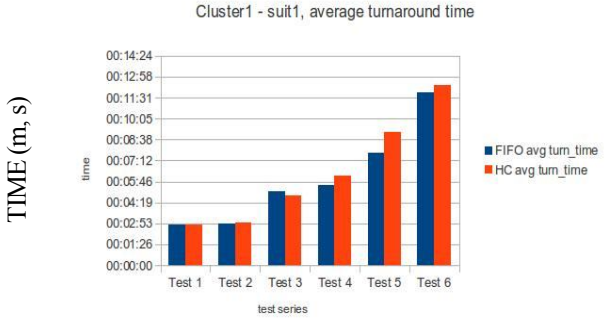

Figure 7 – Average turnaround time for each series

Both these test suites were tested with default FIFO scheduler and proposed Hill Climbing optimization algorithm. The results show that both of these schedulers are able to make equally good scheduling decisions in various situations and hardware configurations.

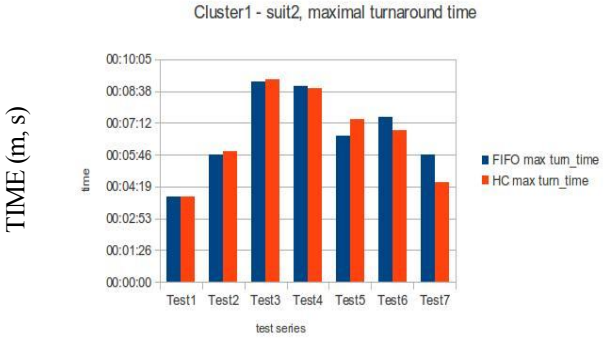

Figure 8 – Maximal turnaround time

CONCLUSIONS. The scheduling algorithm that we implemented can replace original FIFO algorithm released with TORQUE resource manager. It can effectively run user-submitted jobs in acceptable time. One of the disadvantages of this scheduling approach is inability to promptly react to hardware changes in cluster architecture. In case of temporary or permanent node outage scheduler still keeps the old schedule in memory until the request for next job comes. Then it gathers information about the cluster nodes and performs new

schedule generating cycle if necessary. However internal structures of the scheduler allow easy expansion with new, maybe more efficient heuristic algorithm. Another advantage of scheduling-based approach over queue-based is that user can get easily get information about the estimated start time of their job.

REFERENCES

1. Iqbal S., Gupta R., Fang Y. Job scheduling in HPC clusters, Dell Power Solutions, February 2005.

2. Parsa S., Entezari-Maleki R. A queuing network model for minimizing the total makespan of computational grids, // *Computers and Electrical Engineering* 38, April 2012.

3. Jones J.P. *PBS Professional 7, administrator guide*, Altair, April 2005.

4. Xu M. Q. Effective metacomputing using LSF multicluster, // *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pp. 100–105. IEEE, 2001.

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

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

7. Adaptive Computing Enterprises, Inc. *Moab workload manager administrator's guide, version 6.1.4*, February 2012 http://docs.adaptivecomputing.com/

8. Sotskov Yu.N., Shakhlevich N.V. NP-hardness of shop-scheduling problems with three jobs // *Discrete Applied Mathematics* 59 (1995) 237–266.

9. Duvivier D, Preux P, El-Ghazali Talbi. Climbing Up NP-Hard Hills. //*Parallel Problem Solving from Nature IV*, pages 574–583, Springer, 1996.

ВНЕДРЕНИЕ АЛГОРИТМА ОПЕРАТИВНОГО ПЛАНИРОВАНИЯ ВЫСОКОПРОИЗВОДИТЕЛЬНОГО ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА

Дж. Скринарова, М. Крнак

Университет Матея Бела, Факультет естественных наук

ул. Тайовского, 40, Банска Быстрица, 97401, Словакия. E–mail: jarmila.skrinarova@umb.sk

П. Мартинкова

Жилинский университет, Факультет науки управления и информатики

Университет, 8215/1, Жилина, 010 26, Словакия. E–mail: krtcom@gmail.com, Penka.Martincova@fri.uniza.sk

Представлен разработанный авторами алгоритм планирования заданий высокопроизводительного вычислительного кластера. Взятая за основу методика планирования базируется на эффективном и широко распространенном алгоритме локального поиска (Hill Climbing). Авторами проведен сравнительный анализ двух подходов к планированию заданий в параллельных системах: в порядке очередности и в порядке задания. Предложенный авторами алгоритм был задействован в менеджере ресурсов TORQUE в условиях реального производства.

Ключевые слова: планирование высокопроизводительных вычислений, планирование заданий, планирование в порядке задания.

Стаття надійшла 13.03.2013.