DOI: https://doi.org/ 10.31341/jios.43.2.5
Open Access
UDC 004.414.23
Original Scientific Paper

Parallel Simulation of Tasks Scheduling and Scheduling Criteria in High-performance Computing Systems

Jarmila Skrinarova *jarmila.skrinarova@umb.sk*
Faculty of Natural Science
Matej Bel University, Banská Bystrica, Slovakia

Michal Povinsky *michal.povinsky@umb.sk*
Faculty of Natural Science
Matej Bel University, Banská Bystrica, Slovakia

Abstract

This work is focused on the issue of job scheduling in a high performance computing systems. The goal is based on the analysis of scheduling models of tasks in grid and cloud, design and implementation of the simulator on the base of GPGPU. The simulator is verified by our own proposed model of job scheduling. The simulator consists of a centralized scheduler that is using GPGPU to process large amounts of data by parallel way. In order to ensure the optimization of the scheduling process we have implemented a simulated annealing algorithm. GPGPU model was compared to the CPU when the number of nodes from 32 to 2048. Improving the implementation based on GPGPU had a significant impact on the system with 512 nodes and with an increasing number of nodes further accelerates in comparison with sequential algorithm. In this work are designed new scheduling criteria which are experimentally evaluated.

Keywords: HPC, grid computing, cloud computing, hybrid computing, elastic cluster, management of resources, job scheduling, virtual machine scheduling

1. Introduction

Cloud computing is now very popular and is mainly oriented to commercial purposes. High performance computing has traditionally been done on supercomputers and high-performance clusters, often related to computational grids. In this work we have tried to contribute to a gradual transition from the grids, as relatively fixed network clusters to Grid-based elastic clusters, which elasticity and ability to lease resources from the external environment leads to the creation of a virtual computer world. The philosophy of this work is focused on the virtual machine, which is dedicated mainly to scientific computations and which is able to elastically expand or reduce according to the dynamic load of the system. The elastic cluster presents a hybrid computing, which uses appropriate grids and cloud

properties. Cooperation of private and public clouds contributes to the key objective of the quality of services [1].

An elastic cluster represents a unified model of managing HPC (High Performance Computing) resources and cloud resources. For example, the system OpenNebula is designed to manage a group of virtual machines. The Elastic clusters contain comprehensive solutions that provide services for management of dynamic infrastructure, infrastructure on the cluster level and interconnection between the dynamic and cluster infrastructure.

The elastic cluster supports both virtual and physical resources. Using virtual machines in the cloud allows efficient running of HPC jobs. Elastic clusters have to satisfy the quality criteria related to the latency caused by assignment of virtual resources. Amazon Elastic MapReduce (Amazon EMR) is a web service that allows you to handle large jobs with solid scientific data on dynamic web infrastructure using Amazon Simple Storage Service (Amazon S3).

In summary, the contributions of this work are the following: We design and create new job scheduling simulator on the base of GPU that takes into account physical and virtual resources of hybrid system. We implemented and verified it on real HPC cluster environment that contains physical and virtual machines. We optimize the schedule by simulated annealing algorithm. The aim is to optimize the schedule based on the optimization criteria, the time of creation schedule, the completion time of the last task $C_{max..}$ We designed two new criteria. New criteria are based on energy functions. The first energy function helps us to effective stop optimizing algorithms. The second energy function is complex criterion for evaluation quality of schedule. We designed methodology, created and evaluated four experiments with goal to find models responsible for the quality of schedule.

The rest of this paper is organized as follows. Related work is discussed in Section 2. The models of dynamic resource management in computing systems are described in Section 3. Section 4 outlines Scheduling criteria suitable for elastic cluster. In this section we designed two new types of energy function as complex criteria for evaluation quality of schedule. The Compute Unified Device Architecture is introduced in Section 5. Design of job scheduling simulator is described in Section 6. In Section 7 computing environment for the implementation and testing of the scheduling simulator is specified. Section 8 includes testing of the scheduling simulator design and methodology of measurement and scheduling algorithm verification focused on requirement achievement specified in Section 4. This section also describes the obtained scheduling results and their validation. Finally, Section 9 contains the conclusions of this work.

2. Related Works

This paper is based on our previous work related to the proposal of models for parallel tasks in workflow, their computing in a distributed environment, and also from the design of algorithms to support the scheduling [2, 3, 4]. Some approaches of stochastic and heuristic optimization methods for scheduling in grids are suggested. Implementation of genetic algorithms and simulated annealing is

described in [5, 6, 7, 8]. Lim et al. [9] proposed a hierarchical parallel genetic algorithm (Eng. Grid-Enabled Hierarchical Parallel Genetic Algorithm, GE-HPGA). Other methods are based on the local search heuristics]. Hybrid model of genetic algorithm and Tabu search suggested Xhafa et al. [10, 11]. Other approaches focus on fuzzy PSO optimization [12], artificial neural networks [13], neural trees [2] and economically based methods [14, 15].

3. Models of Dynamic Resource Management in Computing Systems

In this section we deal with models that are used to manage the workload and resources in dynamic systems. System resources can be represented as computational nodes. Workload is defined by batch of tasks to be performed on certain computing resources using scheduling rules. Resources can be physical or virtual. There are two basic models approach to resource management [16, 17]:

- Model management workload and resources.
- Model the dynamic infrastructure.

The first model is the management of workload and resources. It is called Workload and Resource Management Systems, WRMS. Model WRMS provides three kinds of activities:

- Managing resources, including resource management conditions.
- Managing tasks, including creating, assigning the ranks, waiting and monitoring.
- Scheduling, mapping tasks to a set of resources and the allocation of resources for certain tasks at a time.

This is principle of work for resource management systems such as Oracle Grid Engine [6], Torque and Condor.

The second model is used for dynamic resource management infrastructure (Dynamic Infrastructure Management System, DIMS). These include systems like OpenNebula, xCAT or VMware vCenter Server. The DIMS model supports two types of functions:

- Managing physical resources.
- Managing services, including the creation of services and the subsequent allocation of resources, monitoring services, migration and cancellations. All of these services can manage virtual or physical resources.

Typical tasks in the workflow can include precedence dependences. There is a list of all tasks belonging to the task j that must be completed before task j starts running. Parallel tasks in the workflow contain subtasks that can communicate together. The group of tasks need to be scheduled on resources of a single machine with multiple computational units (a group of processors, gang) to avoid unnecessary waiting. Tasks in the workflow, which use synchronization, cause also waiting subtasks, but may slow down other tasks in the system. The task scheduler system is responsible for finding a suitable group of machines that are able to perform all the subtasks of a task.

4. Scheduling Criteria Suitable for Elastic Cluster

A schedule means assignment of tasks to recourses for a certain time interval so that no two jobs are executed simultaneously on the same resource and resource capacity is not exceeded too. The schedule specifies for each point in time a set of tasks that are performed at that moment on a particular set of resources [1].

S_j denotes to start processing time of task J_j, and C_j is time of task J_j completion. The most commonly used criteria for optimization of the schedule φ, that are needed to be minimized [18]:

- Completion time of the last task Cmax is called the maximal time of end of task (makespan), $C_{max} = max_j\{C_1, \ldots, C_n\}$. For schedule φ we can define completion time of schedule (1):

$$C_{max}(\varphi) = max_j\{C_j\}. \quad (1)$$

The criterion deserves serious attention, because processing time represents the entire input set of tasks, and thus the length of the schedule.

- The total completion time (flowtime) of all jobs is calculated as sum of all the tasks $F = \Sigma\ C_j$ for all j from 1 to n. If the tasks have different weights we calculate a weighted sum of all tasks $\Sigma\ w_jC_j$. For schedule φ we can define flowtime of schedule (2):

$$F(\varphi) = \Sigma\ C_j \quad (2)$$

Based on the criteria and (2), we propose two types of energy function (3) and (4)

$$E_1(\varphi) = n*C_{max}(\varphi)+F(\varphi). \quad (3)$$

$$E_2(\varphi) = a_1*n*C_{max}(\varphi) + a_2*F(\varphi) + a_3*TC, \quad (4)$$

where a_1, a_2, a_3 are weights coefficients and TC are transfer costs. Transfer costs are caused mainly by transfer files and creating virtual machines.

In our case, we use several evaluation criteria and restrictive conditions for an elastic cluster.

If r_j is the time availability of task j, S_j is time to start processing task j, C_j is competition time of task j, T_jS is the last time for starting task j, D_j is duration of task j, that is estimated by client, then we can define evaluation criteria for elastic cluster.

The elastic cluster typically supports three types of time requirements of task, which define:

- Limit for maximum slowdown of task SD_{max}.
- An advance reservation of resources.
- Last possible time T_jS (deadline) for starting the task j.

By equation 5 we calculate slowdown of tasks for the tasks that have required a limit to the maximum slowdown of task SD_{max}.

$$SD(j)=(S_j - r_j + D_j)/D_j. \quad (5)$$

We guarantee that the slowdown does not exceed the value SD_{max}. Using equation 6 the latest start time of the task can be calculated.

$$T_j^S = r_j + D_j(SD_{max}-1). \quad (6)$$

5. Compute Unified Device Architecture

In this section Compute Unified Device Architecture (CUDA) is introduced. To perform computation using CUDA it is necessary to define a C function, called the kernel. This function uses specified number of lightweight threads. Threads are grouped in blocks, and blocks form a grid. This hierarchy is based on the hardware organization of GPU.

Threads within a block can easily communicate through shared memory. Blocks of threads can run independently. The GPU hardware contains a number of processors, and each block can be executed on a different processor. The order in which blocks are dispatched to processors is managed by the embedded scheduler. There may be a number of blocks being executed on a processor at the same time, as long, as sufficient resources are available (e.g. shared memory, registers). There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same processor core and must share the limited memory resources of that core. Currently, the CUDA specification states that each block may contain up to 1024 threads, organized in 1-, 2- or 3-dimensional array. The number of thread blocks in a grid is usually specified by the size of the data being processed or the number of processors in the system.

CUDA threads may access data from multiple memory spaces during their execution. Each thread has private local memory. Each block of threads has shared memory visible to all threads of the block and with the same lifetime as the block. All threads have access to the same global memory.

There are two additional read-only memories accessible by all threads: the constant and texture memory spaces. The global, constant, and texture memory spaces are optimized for different memory. For a more detailed description, please see to the CUDA Programming Guide by NVidia [19].

6. Design of Job Scheduling Simulator

We have created simulator for job scheduling in high performance computing systems. The simulator is written in C language. We decided to write own simulator as an experimental environment that we can fill in or change. Internal structure is based on matrices:

- NODE_LATENCY – communication latency between machines.
- CPU_SPEED – processor speed for every machine.
- JOB_LENGTH – job time consumption in number of operations.
- FILE_SIZE – capacity of files.

- FILE_MIRRORS – time of creating copy of file on the node (0 if file is on the particular node, -1 if file isn't on the node).
- JOB_FILES – list of input files for every job.
- FILE_GENBY – for every job includes list of files which are generated by this job.
- NODE_BANDWIDTH – describes bandwidth between machines in the system.
- NODEJOBS – list of jobs actually allocated on the machine.
- NODEJOBCOUNT – numbers of job allocated on the machines.
- NODECOMPLETEDJOBS – numbers of completed jobs for every machine.
- NODELASTJOBFINISH – time of completion last job on every machine.
- JOBDEP – contains dependences between jobs.
- JOBFILESLIST – contains actual list of files for every job in process.
- JOBFILESCOUNT – number of files for every job.
- NODEJOBSBEST – copy of matrix NODEJOBS which contains the best times.
- NODEJOBCOUNTBEST – copy of matrix NODEJOBCOUNT which contains the best times.
- NODEEDOWNLOADTIMES – time of completion the last file transfer on the machine.
- NODEFILEDOWNLOADFOR – the job file downloaded for.
- JOBFINISH – completion time of jobs.
- KERNEL_STATUS – output file from GPU kernel.

Input consists of two parts:
First part of input consists of basic information about scheduled system:
- Number of nodes.
- Number of machines.
- Number of jobs.

Second part of input is a generated set of inputs matrix and contains information:
- Connection characteristics between machines.
- Speed of processors.
- Lengths of jobs.
- Size of files.
- List of input files of jobs.
- List of files which are created by every job.
- Start allocation of files.

In the process of simulation output from simulator is calculated. The output contains:
- Completion transfer time of every needed file.
- Start time of every job running.
- Completion time for every job.

- The output from the simulator is in the same time the input for the optimizing algorithm.

We have prepared two versions of simulator. The simulator of job scheduling can run as:

- A sequential program on CPU.
- A parallel program on GPU.

In parallel version all matrices are stored in global memory. We used threads and block of threads in the program. One thread in program is responsible for creating schedule on one machine. Blocks are consisted of 16 threads.

Basic algorithm of GPU simulator can be described by pseudoalgorithm:

Algorithm 1: Job scheduling GPU simulator

Inputs: number of nodes, number of machines, number of jobs, connection characteristics between machines, speed of processors, lengths of jobs, size of files, list of input files of jobs, list of files which are created by every job, start allocation of files.

Outputs: completion transfer time of every needed file, start time of every job running, completion time for every job.

if not completed all jobs or is found better completion time **then**
for every machine **do** kernel function:
for every job on the machine **do**
for every file of job **do**

1. Search resource machine from whom will transfer file (if it is needed).
2. Compute time for file transfer completion and write it to matrixes NODEEDOWNLOADTIMES a FILE_MIRRORS.
3. Find out time of last transferred file.
4. Compute start time of running job (maximum of completion time of last transfer and completion time of last job).
5. Compute v running time of job and completion time of job.

end for
end for
end for
end if

The output from the simulator is in the same time the input for optimizing algorithm. Simulated annealing (SA) algorithm is responsible for optimizing created schedule of jobs.

Schedule optimizing SA pseudoalgorithm:

Algorithm 2: Schedule optimizing SA algorithm

Inputs: completion transfer time of every needed file, start time of every

Outputs: number of iteration, actual C_{max}, best C_{max}, actual flowtime, best flowtime, actual E_1, best E_1, (difference between actual E_1 and best E_1), probability function, neighbor function and temperature function

Generate random initial solution.
Evaluate current schedule.

Remember current schedule as the best solution.
Repeat for the defined number of iterations:
1. Randomly move jobs.
2. Randomly change file download order.
3. Evaluate the new schedule.
4. Calculate current temperature.
5. **if** the temperature is positive **then** calculate probability of accepting the new schedule. **else** accept the schedule if it isn't worse than the previous schedule. **end if**
6. **if** the schedule has been accepted **then** set the schedule as the best schedule. **else** return to the best solution. **end if**

The algorithm of SA is not difficult to implement. But efficiency of algorithm for reason of particular problem depends on functions:

- Energy function *E,* which represents energy of a given state.
- Neighbor function *N*, which produces the neighbors of a given state.
- Probability of solution acceptance function *P*, that determines whether moves should be accepted or not.
- Temperature function *T*, which computes the cooling schedule.

Investigation of the first function is one goal of our experiments, because may cause difficulties in implementation depend on the problem specification. See equations (3) and (4).

7. Computing Environment for the Implementation and Testing of the Scheduling Simulator

The proposed scheduling simulator is implemented and tested on high-performance cluster that is allocated in Matej Bel University in Banska Bystrica, Slovakia as framework of Slovak Infrastructure for High Performance Computing (SIVVP) project. The main objective of SIVVP is the creation of a Slovak grid and supercomputing infrastructure in some centers. Specialized centers are gradually equipped with modern technologies for high-performance computing, supercomputing and high-capacity data storages. The equipment is situated in Slovak academy of Sciences in Bratislava and in 4 universities in Slovakia: Slovak Technical University in Bratislava, University of Zilina, Technical University in Kosice and Matej Bel University in Banska Bystrica. Infrastructure of local sites is interconnected on the base of rules of National Grid Initiatives and European Grid Initiatives. Development and implementation of new technologies shows the need to focus Slovak infrastructure towards hybrid computing. This means to search and apply appropriate services, typical for cloud computing and the main advantages supercomputer clusters and grid computing that can be named as an elastic cluster.

Currently, UMB cluster consists of 49 servers, each server has 12 or 16 computing elements (560 total), 48 - 128 GB of RAM per server and 96 TB of total data storage capacity. The cluster contains 2 special graphics accelerators nVidia

Tesla M2070 with 448 computing cores per card and 6 nVidia Tesla K20 with 2496 computing cores per card which are designed for general computing GPGPU. The system includes 2 servers for management, system security and data access, 2 data storages and high-speed Infiniband network between computing nodes and data storage arrays.

We use Scientific Linux 6.2 operating system on all computing nodes in the cluster. Part of the software installation is xCAT (Extreme Cloud Administration Toolkit) designed to manage computing nodes in the cloud environment and IBM DS Storage Manager that is intended to manage storage arrays.

The High Performance Computing Centre – HPCC was established in the context of a national project SIVVP at Matej Bel University, see www.hpcc.umb.sk . The main objective of the work of Centre is to provide services to scientists who, in their research work need to use high-performance computing environment.

8. Testing of the Scheduling Simulator Design and Methodology of Measurement and Scheduling Algorithm Verification

We prepared two main schedulers for experiments. First scheduler is responsible for creating the schedule by sequential way and runs on CPU. The second one is based on parallel program and uses GPUs. We prepared generated data for experiments. The prepared data have the structure typical for high performance computing system and job characteristics. We designed two load system models. The load system model #1 represents batch of small load jobs for high performance computing. In the load system model #1 each machine is in average loaded by 4 jobs and 16 files. For example, for system with 32 machines it means that load do not exceed 128 jobs for high performance computing which use 512 files.

The load system model #2 represents batch of big load of high performance computing jobs. In the load system model #2 each machine is in average loaded by 16 jobs and 32 files. For example, it means that for system with 1024 machines load do not exceed 16384 jobs for high performance computing which use 32768 files. Experiments were done for systems with 32, 64, 128, 256, 512, 1024 and 2048 physical or virtual computing machines with different processor speed, differences between machines, bandwidth and so on.

The first goal of the experiments is to investigate and compare times needed for creating optimized schedule for both sequential and parallel scheduling algorithms on different high performance computing systems for both load system models. Optimization algorithm was stopped after a fixed number of iterations (1000 and 5000).

We can see in table 1 and on the figure 1 that for small load systems with machines 32 – 256 is better to use sequential job scheduling program. But for more than 512 machines is better to use parallel scheduling. In system that contains 1024 machines the parallel created schedule reached more than 4 – times improving. It means that parallel schedule creation process takes only 22.6 % of time then sequential one.

Numbers of machines	Time of schedule creating on CPU [sec.]	Time of schedule creating on GPU [sec.]
32	2	29
64	5	60
128	21	144
256	108	344
512	1468	806
1024	8782	1989

Table 1. Time of schedule creating on CPU and GPU for load system model #1 and different numbers of machines.

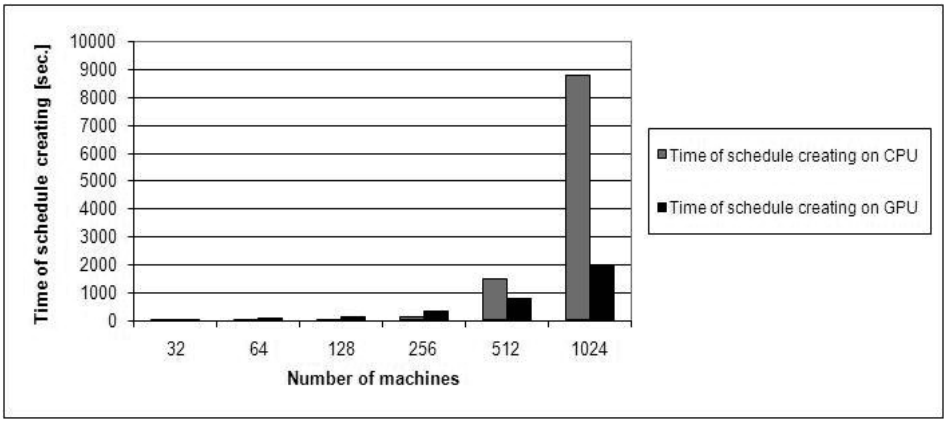

Figure 2. Comparison of times of schedule creating on CPU and GPU for load system model #1 for different numbers of machines.

Numbers of machines	Time of schedule creating on CPU [sec.]	Time of schedule creating on GPU [sec.]
32	13	184
64	77	298
128	979	751
256	8046	1508
512	N	3654
1024	N	7706

Table 2. Time of schedule creating on CPU and GPU for load system model #2 and different numbers of machines.

In the table 2 and figure 2 we can see results for big loaded high performance systems. For systems with 32 – 64 computing machines is convenient to use sequential scheduling. For systems with more than 128 machines is needed parallel scheduling. Sequential program did not finished computing of the schedule for 512 and 1024 computing machines.

In a system that contains 256 machines the parallel created schedule reached more improving than 5.3 – times. Parallel schedule creation process takes only 18.7 % of time then sequential one.

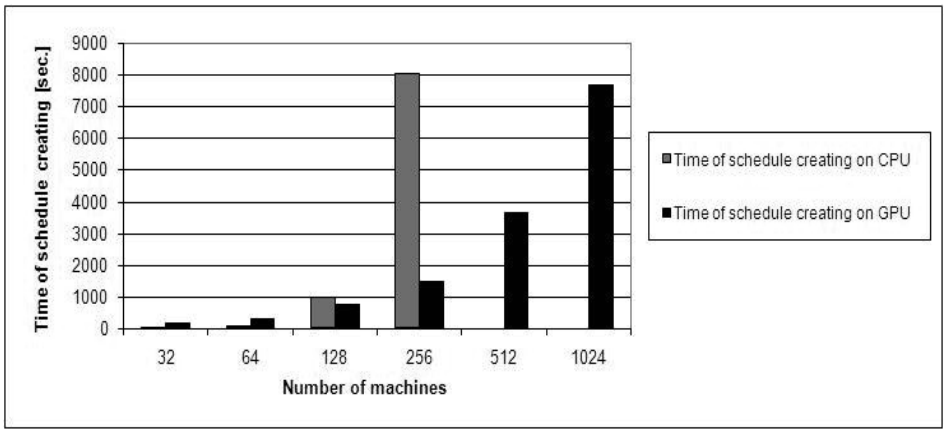

Figure 2. Comparison of times of schedule creating on CPU and GPU for load system model #2 for different numbers of machines.

Numbers of machines	Completion time of last job for load system model #1	Completion time of last job for load system model #2
32	48645	91025
64	3460	8692
128	2880	9033
256	3510	9768
512	4620	13234
1024	5980	14520

Table 3. Time of last job completion C_{max} in the load system model #1 and #2.

The second goal of experiment was to find an optimal (or pseudo optimal) schedule for small and big loaded computing systems. Results we can see in the table 3 and they are specified in numbers of time units.

The parameters of stop the optimizing algorithm were the same for sequential as for parallel creating schedule. So that the results are the same, except the sequential program that was no able in particular time limit compute schedule for 512 and 1024 computing machines and big loaded system data model #2. On the figure 3 and figure 4 are graphically displayed times of completion last job C_{max} in the small load system model #1 and in the big load system model #2. Comparison completion time of last job C_{max} in the small and big loaded system model is on the figure 5.

The third goal of experiment is to investigate and compare times needed for creating optimized schedule for both sequential and parallel scheduling algorithms on different high performance computing systems for both load system models.

Optimization algorithm was stopped if last 50 iterations energy function E_1, see equation (3), is not improving (is not decreasing).

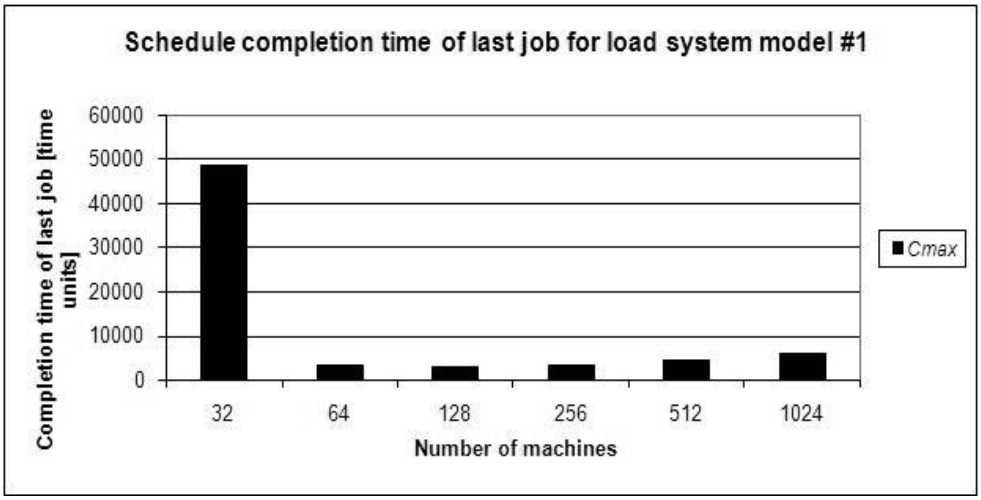

Figure 3. Values of completion time job C_{max} in the small loaded system model #1.

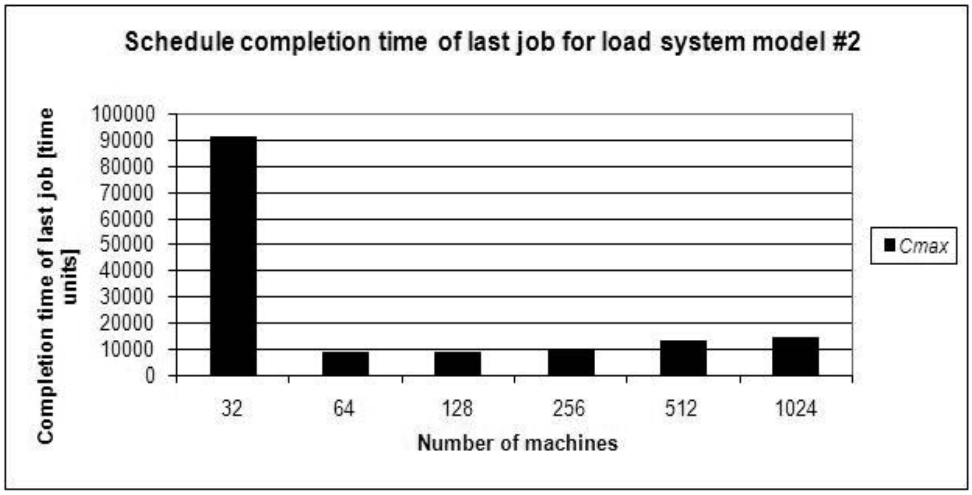

Figure 4. Values of completion time job C_{max} in the small loaded system model #2.

We can see in table 4 that for small load systems with machines 64 – 512 is better to use sequential job scheduling program. But for more than 512 machines is better to use parallel scheduling. In system that contains 2048 machines the parallel created schedule reached more than 4 – times improving. It means that parallel schedule creation process takes only 21.3 % of time then sequential one.

In the table 5 we can see results for big loaded high performance systems. For systems with 64 – 256 computing machines is convenient to use sequential scheduling. For systems with more than 512 machines is needed parallel scheduling.

In system that contains 2048 machines the parallel created schedule reached more than 15 – times improving. It means that parallel schedule creation process takes only 6.6 % of time then sequential one.

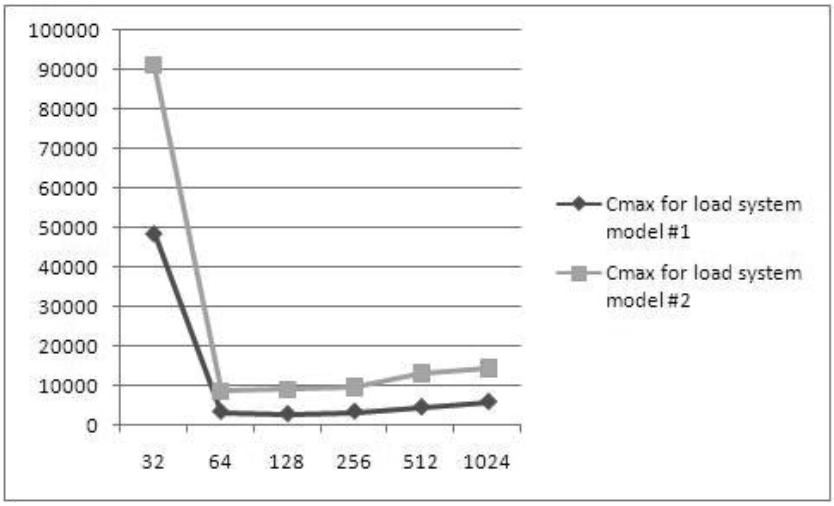

Figure 5. Comparison of last job completion time C_{max} in the small and big loaded system model.

Numbers of machines	Time of schedule creating on CPU [sec.]	Time of schedule creating on GPU [sec.]
64	8.042	95.868
128	34.799	200.03
256	255.67	496.66
512	1777.00	1236.00
1024	14432.00	2565.00
2048	105625.00	7019.00

Table 5. Time of schedule creating on CPU and GPU for load system model #2 after improving stopped criterion of algorithm.

In connection with the use of energy function E_1 (3), we achieved a reduction in the number of iterations and shortening time of schedule creation. On the base of experiment and after analyze measured data we can conclude that average improvement on CPU is 17.79 % and on GPU is 72.21 % for small load system model (table 6). For big load system model is average improvement on CPU 11.67 % and on GPU is 84.49 % (table 7).

The fourth goal of experiment is to investigate our proposal of energy function E_2, see equation (4) as a complex criterion of schedule quality. Energy function E_2 depends on completion time of schedule, number of nodes, schedule flowtime and transfer costs. Weight coefficients we set on a_1=0.35, a_2=0.35, a_3=0.3 and transfer costs on 0, 5, 10 and 25 %. Comparison of energy function E_2 with transfer costs 0, 5, 10 and 25 % for load system model #1 we can see on table 8 and figure 6.

Numbers of machines	Improved number of iterations on CPU	Improved of schedule creation time on CPU [%]	Improved number of iterations on GPU	Improved of schedule creation time on GPU [%]
64	982	3.97	924	8.23
128	996	21.39	771	29.70
256	1049	53.38	1199	0.00
512	1018	19.06	879	13.77
1024	3308	6.05	212	371.70
2048	4253	2.92	954	9.85

Table 6. Improving process for schedule creation for load system model #1.

Numbers of machines	Improved number of iterations CPU	Improved of schedule creation time on CPU [%]	Improved number of iterations GPU	Improved of schedule creation time on GPU [%]
64	1388	15.92	218	358.72
128	1352	11.17	1004	0.00
256	2361	4.49	1036	0.00
512	2438	8.24	1068	4.59
1024	4237	21.27	875	14.29
2048	7080	8.93	436	129.36

Table 7. Improving process for schedule creation for load system model #2.

Numbers of machines	Energy function E_2 with 0% transfer costs	Energy function E_2 with 5% transfer costs	Energy function E_2 with 10% transfer costs	Energy function E_2 with 25% transfer costs
64	24714.56	47690.21	109395.26	259951.37
128	65060.84	127329.11	261197.47	611860.92
256	174838,04	138607,04	202695,94	159212,39
512	610780.30	515638.89	728897.45	591133.18
1024	957908.38	881491.50	696758.58	637495.88
2048	3190675.54	3190014.30	2319272.60	2256150.73

Table 8. Energy function E_2 with coefficients a_1=0.35, a_2=0.35, a_3=0.3 and transfer costs on 0, 5, 10 and 25 % for load system model #1.

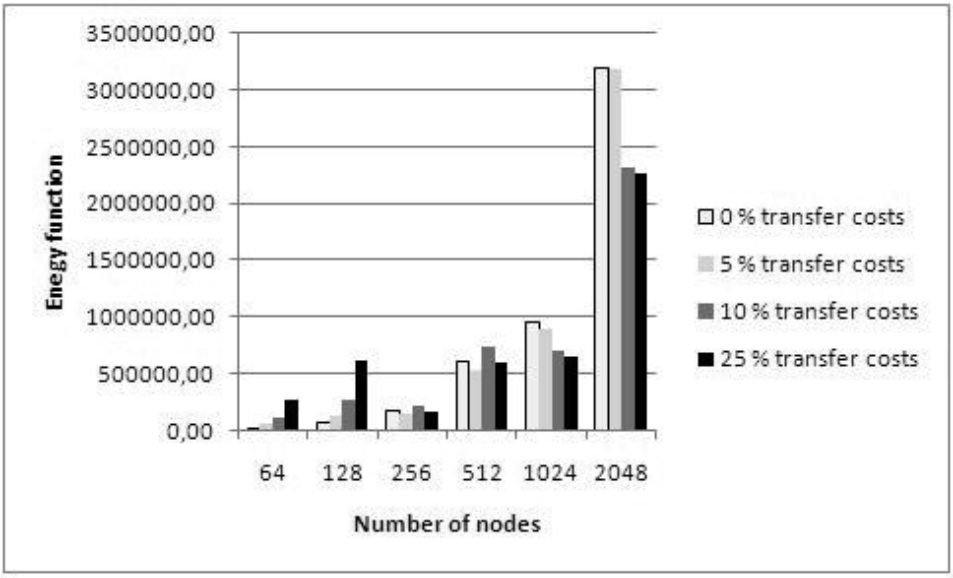

Figure 6. Comparison of energy function E_2 with transfer costs 0, 5, 10 and 25 % for load system model #1.

Comparison of energy function E_2 with transfer costs 0, 5, 10 and 25 % for load system model #2 we can see on table 9 and figure 7.

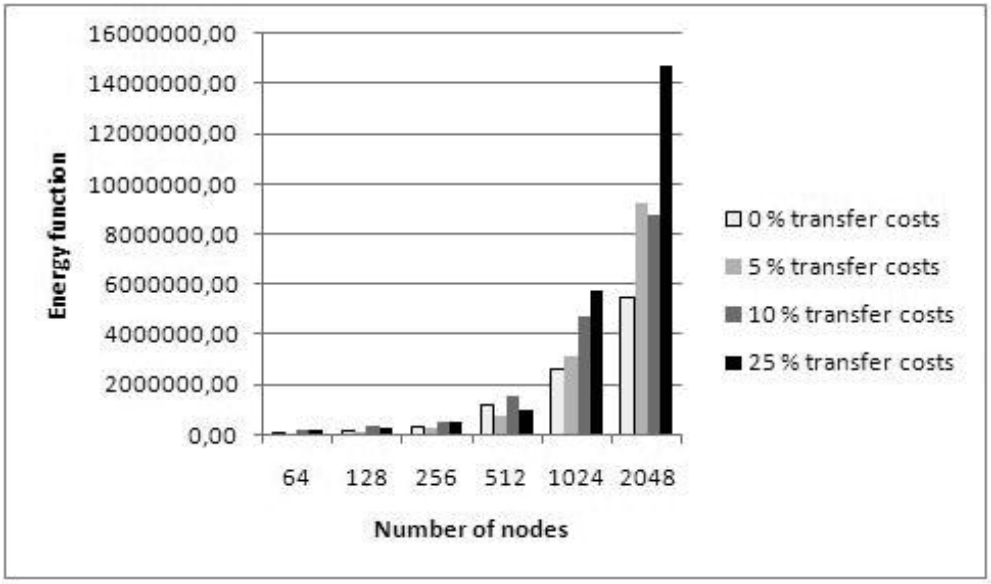

Figure 7. Comparison of energy function E_2 with transfer costs on 0, 5, 10 and 25 % for load system model #2.

Based on evaluation experiments, we can conclude that the GPU job scheduling simulator is effective tool for modeling and evaluation different load system models.

With help of simulator we can investigate various models for heterogeneous or hybrid systems. In the future work is possible to do more precise analysis of virtual machines and their behavior inside of the physical system. Investigation different system models, in context of combination of physical and virtual machines, is very important from point of view managing their images and time of availability for applications.

Numbers of machines	Energy function E_2 with 0% transfer costs	Energy function E_2 with 5% transfer costs	Energy function E_2 with 10% transfer costs	Energy function E_2 with 25% transfer costs
64	63413.48	61460.27	142046.33	136438.66
128	154334.70	142181.14	305987.45	268544.87
256	306087.08	333765.90	488058.46	511702.10
512	1151302.36	762769.83	1486293.45	927528.72
1024	2565776.98	3145666.38	4658539,32	5759775.81
2048	5417199.06	9260877.29	8723311.99	14700214,62

Table 9. Energy function E_2 with coefficients a_1=0.35, a_2=0.35, a_3=0.3 and transfer costs on 0, 5, 10 and 25 % for load system model #2.

9. Conclusion

In this paper we introduced models of dynamic resource management in computing systems and scheduling criteria suitable for elastic cluster. We designed two new types of energy function as complex criteria for evaluation quality of schedule. After brief introduction the Compute Unified Device Architecture and computing environment we designed, implemented a tested of job scheduling sequential and parallel simulator. The new job scheduling simulator is on the base of GPU that takes into account physical and virtual resources of hybrid system. We implemented and verified it on real HPC cluster environment that contains physical and virtual machines. We optimize the schedule by simulated annealing algorithm. The aim is to optimize the schedule based on the optimization criteria, the time of creation schedule, the completion time of the last task C_{max}.. We designed two new criteria. New criteria are based on energy functions. The first energy function helps us to effective stop optimizing algorithms. The second energy function is complex criterion for evaluation quality of schedule. We designed methodology, created and evaluated four experiments with goal to find models responsible for the quality of schedule.

Acknowledgements

Computing was performed in the High Performance Computing Center of the Matej Bel University in Banska Bystrica using the HPC infrastructure acquired in project

ITMS 26230120002 and 26210120002 (Slovak infrastructure for high-performance computing) supported by the Research & Development Operational Program funded by the ERDF.

References

[1] G. Mateescu, W. Gentzsch, G. J. Ribbens, "Hybrid Computing – where HPC mets grid and cloud computing," in Future generation Computer Systems. 27, 2011, s. 440 – 453.

[2] J. Skrinarova, L. Huraj, V. Siladi, "A neural tree model for classification of computing grid resources using PSO tasks scheduling," in Neural networks world. 2013. ISSN 1210-0552

[3] J. Skrinarova., M. Krnac, "Particle Swarm Optimization for Grid Scheduling: in Eleventh International Conference on Informatics, Rožňava. pp. 153-158

[4] J. Skrinarova, M. Melichercik, "Measuring concurrency of parallel algorithms," in 1st International Conference on Information Technology Gdansk : IEEE computer society, 2008, pp. 289-292, IEEE Katalog Number: CFP0825E-PRT. ISBN 978-1-4244-2244-9.

[5] A.Yarkhan, A., J. Dongarra, "Experiments with scheduling using simulated annealing in a grid environment," in: Proc. of the 3rd International Workshop on Grid Computing, 2002, pp. 232–242.

[6] J. Kołodziej, F. Xhafa: "Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population," In: Future Generation Computer Systems, Volume 27, Issue 8, October 2011. pp. 1035-1046.

[7] J. Skrinarova, F. Zelinka, " The grid scheduling," GCCP 2010 6th International Workshop on Grid Computing for Complex Problems, Bratislava, , 2010.pp. 100-107.

[8] R. Schaefer, R., J. Kołodziej, "Genetic search reinforced by the population hierarchy: Foundations of Genetic Algorithms VII". Morgan, 2003, pp. 369–385.

[9] D. Lim, Y. S. Ong, Y. Jin, "Efficient hierarchical parallel genetic algorithms using grid computing," in Future Generation Computer Systems 23 (4).2007. pp. 658–670.

[10] F. Xhafa, J.A. Gonzalez, K.P., Dahal, A. Abraham, "GA(TS) hybrid algorithm for scheduling in computational grids," HAIS, 2009, pp. 285–292.

[11] G. Ritchie, J. Levine, A fast effective local search for scheduling independent jobs in heterogeneous computing environments. Tech. Rep.,

Centre for Intelligent Systems and Their Applications, University of Edinburgh, 2003.

[12] H. Liu, A. Abraham, A. E Hassanien, "Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm," in Future Generation Computer Systems 26 (8) 2010. pp.1336–1343.

[13] J. Skrinarova., M. Krnac, "Particle Swarm Optimization Model for Grid Scheduling," in Second International Conference on Computer Modelling and Simulation, CSSIM 2011. Brno, Czech Republic. pp. 146-153

[14] R. Buyya, M. M. Murshed. "GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for Grid computing," Concurrency and Computation: Practice and Experience.2002. pp. 1175-1220

[15] T. Goyal, A. Singh, A. Agrawal. "Cloudsim: simulator for cloud computing infrastructure and modeling," in International conference on modeling optimisation and computing. Procedia Engeneering (38), 2012. pp 3566-3572.

[16] A.Beloglazov, J. Abawajy, J. and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," in Future Generation Computer Systems. 28, 2012. pp. 755–768

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

[18] J. Skrinarova, "Implementation and evaluation of scheduling algorithm based on PSO HC for elastic cluster criteria," in Central European Journal of computer science. 4 (3) 2014. pp. 191-201.

[19] NVidia, CUDA C Programming Guide http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#ixzz3lKChsNsN