Available online at www.sciencedirect.com
SciVerse ScienceDirect
Procedia Computer Science 9 (2012) 176 – 185
Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Composite Scheduling Strategies in Distributed Computing with Non-dedicated Resources

Victor Toporkov[a,*], Alexey Tselishchev[b], Dmitry Yemelyanov[a], Alexander Bobchenkov[a]

[a] *National Research University "MPEI", ul. Krasnokazarmennaya 14, Moscow, 111250 Russia,*
E-mail: ToporkovVV@mpei.ru, {groddenator, yemelyanov.dmitry}@gmail.com
[b] *CERN (European Organization for Nuclear Research), CERN CH-1211 Genève 23 Switzerland*
E-mail: Alexey.Tselishchev@cern.ch

Abstract

This work presents dispatching strategies based on methods of job-flow and application-level scheduling in virtual organizations of distributed computational environments with non-dedicated resources. Job-flow management is implemented with the set of specific rules for resource usage. Applications are considered as parallel jobs. Strategies are based on economic scheduling models and diverse administration policies inside resource domains (clusters, computational nodes equipped with multicore processors etc.). Methods of priority economic scheduling of global job flows and local-level applications in distributed computations are studied. Job management structures and economic mechanisms for load balancing in distributed environments are considered.

Keywords: scheduling; strategy; resource management; slot; job; batch; task; economic mechanisms

1. Introduction

Distributed computational environments such as Grid have been known for significant efficiency increase in shared computational resource usage and provision of scientific and enterprise communities with solutions for complex computational tasks. However, those who are responsible for setting up Grid infrastructure and economy encounter difficulties while defining policies and strategies for efficient resource management and job scheduling. *A strategy*, for instance, may be a tradeoff between optimal resource load and fulfillment of all the user requirements. Heterogeneity, changing composition, different owners of different nodes whose computing time is partially shared by users turn the organization of a distributed computational environment into an especially difficult problem.

[*] Corresponding author. Tel. +7-495-362-7145; fax: +7-495-362-5506.
E-mail address: ToporkovVV@mpei.ru.

doi:10.1016/j.procs.2012.04.019

Utility Grid [1], multi-agent systems [2] and cloud computing [3] are types of distributed environments where usage of *economic mechanisms* is seen as promising. Those economic mechanisms are designed to solve tasks like resource management and scheduling of user jobs in a transparent and efficient way. Within the context of any used economic model the interests of different participants of a distributed computing environment (such as end-users or node owners) are often contradictory. It is assumed that node owners may have local job flows (their own tasks) and global job flow (which is formed by external user jobs) competing for limited computational resources of the node. Elaboration of pricing rules which are used to calculate a fee for node computing time usage and take into account user-required quality of service (QoS) is also a very serious problem [1-3]. An overview of various approaches to this problem is given in [4]. Heuristic algorithms that are responsible for resource selection based on user-given utility function are described in [5]. Some resource management models offer simple search and selection of resources required by a user [6] and do not support any optimization. Others do not take into account features related to global and local job competition, the competition among users and other characteristics of distributed environments with non-dedicated computational resources [7]. When constructing a computing environment based on the available resources, e.g. in the model which is used in X-Com system [6], one normally does not create a set of rules for resource allocation as opposed to constructing clusters or Grid-based virtual organizations. This reminds of some techniques, implemented in Condor project [8]. Non-clustered Grid resource computing environments are using similar approach. For example, @Home projects which are based on BOINC system realize cycle stealing, i.e. either idle computers or idle cycles of a specific computer. Another still similar approach is related to the management of distributed computing based on resource broker assignment. Besides Condor project [8], one can also mention several application-level scheduling projects: AppLeS, APST, Legion, DRM, Condor-G, and Nimrod/G. A survey of these systems is given in [9, 10].

A resource broker model [1-5] dynamically employs various economic policies which perform resource management. It is decentralized and application-specific and has two parties: node owners and brokers representing users. Another common trend is related *to virtual organizations* [7, 9, 10] with central schedulers providing job-flow level scheduling and optimization. While former type of resource management is well-scalable, the simultaneous satisfaction of various application optimization criteria submitted by independent users is unreachable in essence and also can deteriorate such integral QoS rates as total execution time of a sequence of jobs or overall resource utilization. The latter type, virtual organizations naturally restrict the scalability. However, scheduling based on uniform and controlled rules for allocation and consumption of resources makes it possible to improve the efficiency of resource usage and find a tradeoff between contradictory interests of different participants.

In this work, we propose two-level model of resource management system which is functioning within a virtual organization (VO). Resource management is implemented with a hierarchical structure consisting of *a metascheduler* and subordinate *job schedulers* that are controlled by the metascheduler and in turn interact with resource managers (e.g., with batch job processing systems). The advantages of hierarchically organized resources managers are obvious, e.g., the hierarchical job-queue-control model is used in X-Com [6], GrADS [11], and Moab scheduler [12]. Hierarchy of intermediate servers allows decreasing idle time for the processor nodes, which can be inflicted by transport delays or by unavailability of the managing server while it is dealing with the other processor nodes. Tree-view manager structure in the network environment of distributed computing allows avoiding deadlocks when accessing resources. Another important aspect of computing in heterogeneous environments is that processor nodes with the similar architecture, contents, administrating policy are grouped together under the job manager control. The application-level optimization begins when the job-flow level optimization is finished. Such a flexible structure coupled with complex metascheduling approach enables multiaspect resource management and makes possible to control dynamic priority of job execution, resource selection and provide multicriteria optimization both on the job-flow scale and for a specific job, according to its requirements and optimization criteria. In some applications jobs require co-scheduling and resource co-allocation on several resources [13-16]. In this case resource allocation has a number of substantial specific features caused by autonomy, heterogeneity, dynamic content changes, and node failures [6, 7, 9, 10].

The proposing approach is similar one in gLite Workload Management System [17], where Condor [8] is used as a scheduling module. But the significant difference between the approach proposed in this work and well-known scheduling solutions for distributed environments such as the Grid [1, 3-7, 17, 18] is the fact that the execution strategy is formed on a basis of formalized efficiency criteria, which efficiently allows to reflect economic principles of resource allocation by using relevant cost functions and solving a load balance problem for heterogeneous processor nodes. At the same time the inner structure of the job is taken into account when the resulting schedule is formed. Thus, two approaches are uniquely combined in a proposed two-tier scheduling model.

This work is organized as follows. Section 2 overviews model components and metascheduling workflow. In section 3 a strategy search is formalized. Section 4 contains simulation results. Section 5 summarizes the work and describes further research topics.

2. Model components

Let us define basic model components presented in this work. Resource is defined as an abstract computational entity, which can be used for execution of one and only one *task*. The complex set of connected interrelated tasks form *a job*. Our model has the following components.

- VO, that defines resource co-allocation dispatching strategies, pricing policies and resource load-balancing mechanisms.
- Heterogeneous hierarchical computational environment that contains computational resources (Grid nodes, CPUs or others) with different performance indices. Each resource is considered as non-dedicated.
- Metascheduler, which implements resource management strategies and policies of VO.
- Application-level schedulers that analyze internal job structure and schedule single tasks.

Each computational node of the heterogeneous environment is mapped to a computational *resource line* in the metascheduler resource management routine. Several resource lines are combined into a virtual resource domain. Each resource line has two static attributes which are its performance P and its base price tag F for a computing time unit. The performance is an inherent parameter of a node and the base price tag is assigned by its owner. The dynamic characteristic of a node is represented with its local schedule which is a list of slots available for reservation. This list is sent to metascheduler by request. A slot is a continuous interval of time and is described with three parameters: its start time, its length and its fee [13-15]. A resource request is a set of a few constraints determined by a user which correspond to the properties of the respective user job. They include: minimal performance requirement for computational nodes, $P_{\min}$; maximal price tag for a single timeslot, $F_{\max}$; number n of simultaneously reserved timeslots; minimal slot length; the internal structure of a job as a directed acyclic graph (DAG), where vertices represent single tasks and edges represent data dependencies [16]; deadline for the job execution. A job may require more than one timeslot if it includes several segments that can be executed in parallel way, for instance. Then the user specifies the number of reserved timeslots and minimal performance requirement that applies for them all. The whole job budget is determined by the timeslot number and the maximum price per timeslot. The minimal timeslot length requires an additional explanation. This is the minimal time estimated by the user which is required to complete job execution given the performance of the nodes meet the minimal requirement $P_{\min}$.

The hierarchical model of the computational environment implies two-tier scheduling (Fig. 1). On the job-flow level the set of independent jobs is distributed between resource domains according to dispatching strategies and economic criteria. Schedule on this level is defined by a metascheduler as a slot set for each job, which is optimal in terms of a whole job set. Application-level schedulers receive the list of resources which were meant to execute the job on and a strategy, which defines the rule used to execute tasks of a concrete job. On this level an optimal slot and specific resource are defined for each single task in a job, thus, making it possible to take internal job structure into account. The metascheduler works in cycles which are quanta of its process. For each cycle it has following information: a set of resource lines and a global job queue. What it needs then is a batch of jobs which is a ranked job list and a subset of available slots for a specific virtual resource domain and a certain timeframe which is called a scheduling interval. The length of the batch and the scheduling interval are parameterized by VO administrators. Jobs are fetched into the batch accordingly to several variables, such as the maximum price tag, deadline, and the number of failed scheduling attempts for a job. These variables being weighted and added up determine job rank according to which it takes a position closer to head or tail of a batch. The preparation phase ends and the actual scheduling process is executed as follows (see Fig. 1). The metascheduler analyzes available slots and finds an optimal slot combination to accommodate every job in a batch using economic criteria. The budget and the deadline defined by the end-user are considered during this step. The algorithms for this step were detailed in [13-15]. After the domain is determined metascheduler defines the strategy for each job. For example as shown on Fig. 1, the user, who has sent the job *i* has the higher budget than the one who has sent the job *k*. The strategy for *i* may be expressed as "*execute as soon as possible*" while the strategy for *k* may be expressed as "*execute as late as possible within the defined deadline*". These jobs are later sent to application-level schedulers and the application-level scheduling begins. Application-level schedulers query internal schedules for all the resources which were selected for each job, analyze the job DAG and form a resulting schedule for every task according to the strategy. Application-level schedulers are guaranteeing that there are no collisions between the tasks which were scheduled and local tasks, which may have priority over the job-flow.

3. Formalization of job-flow and application-level scheduling

Let us note a global resource set $R_g = \{r_p, p = 1,..,M\}$, which includes all resources. A global job-flow is a set of jobs received by the metascheduler in time: $FL_g = \{l_i, c_i, T_i, G_i, i = 1,..,I\}$, where the job i is represented as l_i – the amount of resource slots required, c_i - the maximal budget end-user is ready to allocate for execution of the job, T_i – deadline, G_i – the job DAG. Metascheduler at any time moment may query each resource, receive its local schedule and build a set of slots S_{gt} – idle time intervals.

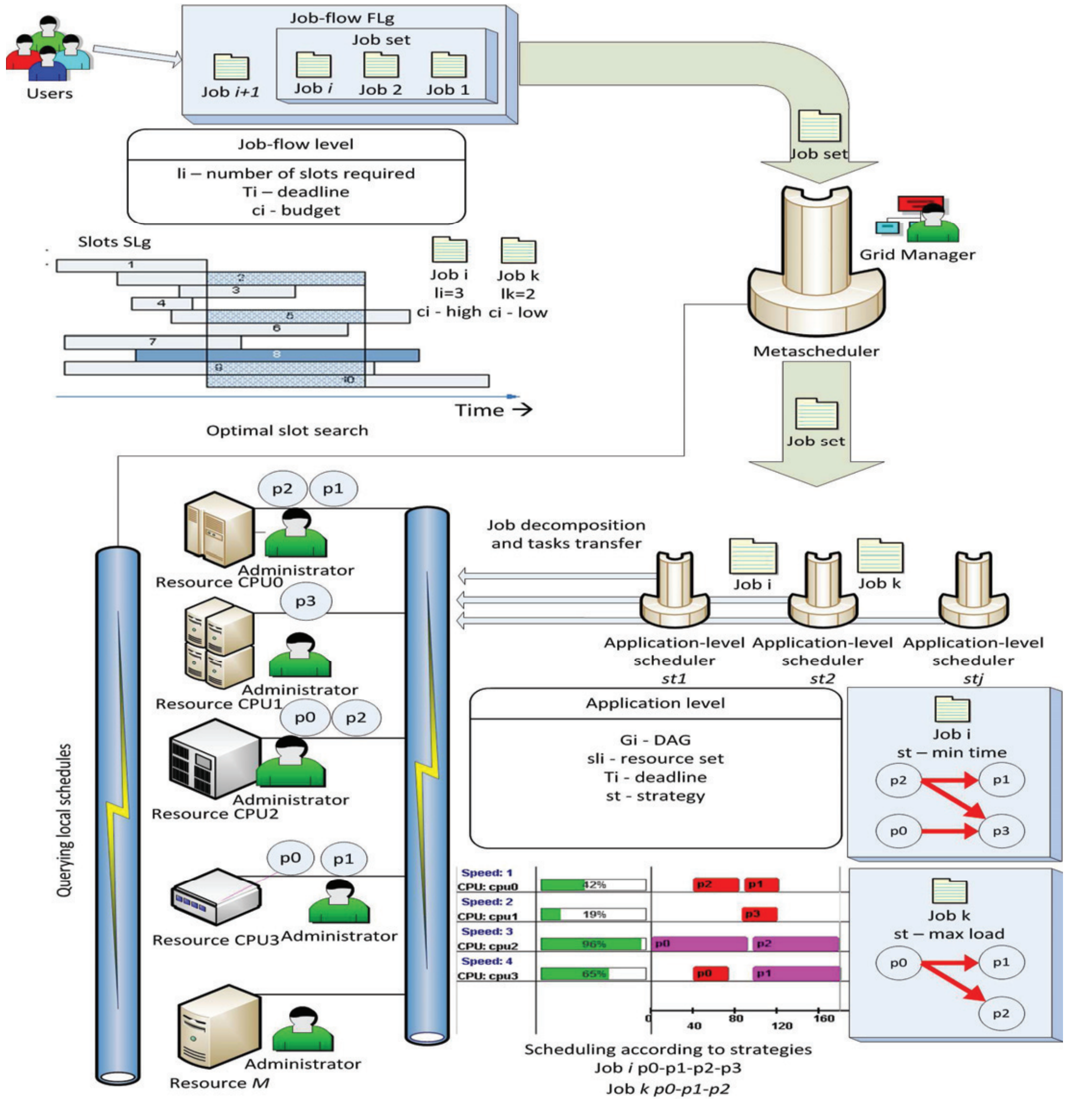


Fig.1. Model components

Let us introduce a set of strategies $ST = \{st_l, l = 1,..,L\}$, which are based on economic criteria and are defined by Grid-managers and developers. Let SL be a set of K slots suitable to execute a subset of jobs $FL_p \subseteq Fl_g$. A slot set is considered as suitable for the job i if the execution is possible in terms of the resource number, the budget c_i and the deadline T_i. It is assumed that for every job there is at least one suitable slot set $sl_i \in SL, sl_i = k, k \in \{1,..,K\}$. On a job-flow level for each job metascheduler aims at finding a slot set sl_i and a strategy st_i for which the value of the function $g_i(sl_i)$, that defines whether the slot set is being effective for the job i, would be optimal [14]. The internal job structure G_i is not taken into account at this time. The mechanism to define $g_i(sl_i)$ which was developed in the previous works [13-15] is now improved. According to the resource request it is required to find a "window" with the following description: n concurrent time-slots providing resource performance rate at least P and maximal resource price not higher than $F_{\max}$ should be reserved for a time span T_i. The length of each slot in the window is determined by the performance rate of the node on which it is allocated. Thus as a result we have a window with a "rough right edge" (Fig. 2). In addition, the criterion of selecting the most suitable set of slots could be specified. This could be the minimum cost, the minimum runtime or, for example, the minimum power consumption criterion. The window search is performed on the list of all available slots sorted by their start time in ascending order. This condition is necessary to examine every slot in the list and for operation of search algorithms of linear complexity [13-15].

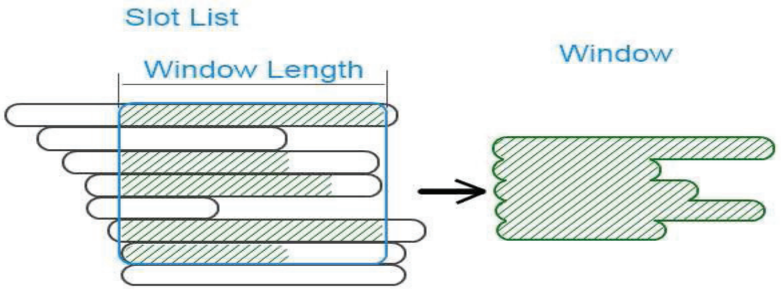


Fig. 2. Window with a "rough right edge"

The scheme of a search for a window that meets the requirements and effective by the given criterion can be represented as follows.

1°. From the list of available system slots the next suitable slot s_k is extracted and examined.

Slot s_k suits, if following conditions are met:

a) Resource performance rate $P(s_k) \geq P$ for slot s_k;

b) Slot length (time span) is enough (depending on the actual performance of the slot's resource) $L(s_k) \geq T_i * P(s_k)/P$.

If conditions **a)** and **b)** are met, the slot s_k is successfully added to the window list.

2°. A current window start time is a set equal to the start time of the last added slot.

3°. Slots whose length has expired considering new window start time T_{last} are removed from the list. The expiration means that remaining slot length $L'(s_k)$, calculated like shown in **step** 1°**b**, is not enough assuming the k-th slot start is equal to the last added slot start: $L'(s_k) < (T_i + (T_{\text{last}} - T(s_k)))P(s_k)/P$, where $T(s_k)$ is the slot's start time. Any combination of the remaining slots can form a window of necessary length.

4°. If the number of slots m in the current window is greater or equal to n, it is required to select n slots, effective on the specified criteria and at the same time satisfying the total cost and deadline restrictions. Suppose the window W of size n with a target criterion value equal to crW was selected. (The problem of selecting efficient window consisting of n slots in the case of $m > n$ will be described below.)

5°. The target criterion value crW of window W is compared with the cr' – the current best target criterion value for all previously found windows. If $crW < cr'$ (in case of a minimization problem) the window W announced as a new window-candidate and crW becomes the new best criteria value: $cr' = crW$. Go to **step 1°**.

6°.The algorithm ends after the last available slot is processed. The result of the algorithm is the window-candidate with the best target criteria value.

The described algorithm can be compared to the algorithm of maximum/minimum value search in an array of flat values. The expanded window of size m "moves" through the ordered list of available system slots. At each step any combination of n slots inside it (in case when $n \leq m$) can form a window that meets all the requirements to run the job. The effective on the specified criteria window of size n is selected from this m slots and compared with the results in the previous steps. By the end of the slot list the only solution with the best criteria value will be selected. Consider the problem of selecting a window of size n with a total cost not more than S from the list of $m > n$ slots (in case when $m = n$ the selection is trivial). The maximal budget is counted as $S = Ft_s n$, where t_s is a time span to reserve and n is the necessary number of slots. The current extended window consists of m slots $s_1, s_2, ..., s_m$. The cost of using each of the slots according to their required length is: $c_1, c_2, ..., c_m$. Each slot has a numeric characteristic z_i the total value of which should be minimized in the resulting window. Then the problem could be formulated as follows: $a_1 z_1 + a_2 z_2 + ... + a_m z_m \rightarrow \min$, $a_1 c_1 + a_2 c_2 + ... + a_m c_m \leq S$, $a_1 + a_2 + ... + a_m = n$, $a_r \in \{0,1\}, r = 1,...,m$. Additional restrictions can be added, for example, considering the specified value of deadline. Finding the coefficients $a_1, a_2, ..., a_m$ each of which takes integer values 0 or 1 (and the total number of '1' values is equal to n), determine the window with the specified criteria extreme value. Job-flow level scheduling ends here.

Application-level schedulers receive following input data.

- The optimal slot set sl and the description of all corresponding resources: $R = \{r_j, j = 1,.., J\} \subseteq R_g$.
- The directed acyclic information graph $G = \{V, E\}$, where $V = \{v_i, i = 1,..n\}$ is a set of vertices that correspond to job tasks, for each of those execution time estimates τ_{ij}^0 on each of resources in R are provided, E is a set of edges that define data dependencies between tasks and data transfer time intervals.
- The dispatching strategy st , which defines the criterion for a schedule expected
- The deadline T_i or the maximal budget c_i for the job (depends on a dispatching strategy and $g_i(sl_i)$).

The schedule which is being defined on an application level is presented as follows: $Sh = \{[s_i, f_i], \alpha_i, i = 1,..,n\}$, where $[s_i, f_i]$ is a time frame for a task i of a job and α_i defines the selected resource. Sh is selected in the way that the criterion function $C = f(Sh)$ achieves an optimum value. The *critical jobs method* [16] which is used to find the optimal schedule and to define f consists of three main steps: 1) forming and ranging a set of critical jobs (longest sets of connected tasks) in the DAG; 2) consecutive planning of each critical job using dynamic programming methods; 3) resolution of possible collisions. A detailed algorithm description is presented in [19].

4. Simulation results

The two-tier model described in the sections 2 and 3 was implemented in a simulation environment on two different and separated levels: on the job-flow level, where job-flows are optimally distributed between resource domains and on the application level, where jobs are decomposed and each task is executed in an optimal way on a selected resource.

4.1. Job-flow level scheduling simulation results

Job-flow level metascheduling was simulated in a specially implemented and configured software that was written to test the features of the two-tier resource management. An experiment was designed to compare the performance of our job-flow level metascheduling method with other approaches such as FCFS and backfilling [12, 20]. Let us remind that our scheduling method detailed in works [13-15] involves two stages that backfilling does not have at all, namely, slot set alternative generation and further elaboration of specific slots combination to optimize either time or cost characteristic for an entire job batch. Backfilling simply assigns “slot set” found to execute a job without an additional optimization phase [20]. This behavior was simulated within our domain with random selection from an alternative slot, each job having one or more of them. So two modes were tested: with optimization (“OPT”) and without optimization (“NO OPT”).

The experiment was conducted as follows. Each mode was simulated in 5000 independent scheduling cycles. A job batch and environment condition was regenerated in every cycle in order to minimize other factor influence. A job batch contained 30 jobs. Slot selection was consistent throughout the experiment. If a job resource request could not be satisfied with actual resources available in the environment, then it was simply discarded. For optimization mode as well as for no-optimization mode four optimization criteria or problems were used: 1) maximize total budget, limit slot usage; 2) minimize slot usage, limit total budget; 3) minimize total budget, limit slot usage; 4) maximize slot usage, limit slot budget.

Results presented in Table 1 apply for the problem 1. As one can see optimization mode, which is using additional optimization phase after slot set generation wins against random slot selection with about 13% gain in the problem 1 whose concern is about maximizing total slot budget thus raising total economical output per cycle and owners' profits.

Table 1. Experimental results for the problem 1: Total budget maximization with limited slot usage

Mode	Average jobs being processed per cycle (max 30)	Average total slot cost per cycle, *cost units*	Average total slot usage per cycle, *time units*	Average slot usage limit per cycle, *time units*
OPT	20.0	11945.98	421.22	471.14
NO OPT	20.0	10588.53	459.36	471.85

Comparable results were obtained for other problems which are summarized in Table 2. Optimized values are outlined in light grey.

Table 2. Experimental results for the problems 2-4

Mode	Average jobs being processed per cycle (max 30)	Average total budget (slot cost) per cycle, *cost units*	Average total slot usage per cycle, *time units*	GAIN, %
Problem 1: Maximize total budget, limit slot usage				
OPT	20.0	11945.9	421.2	+12.8
NO OPT	20.0	10588.5	459.4	
Problem 2: Minimize slot usage, limit total budget				
OPT	12.4	7980.4	300.9	+10.6
NO OPT	12.4	7830.9	332.8	
Problem 3: Minimize total budget, limit slot usage				
OPT	15.1	9242.4	410.057	+6.2
NO OPT	15.3	9813.9	406.612	
Problem 4: Maximize slot usage, limit total budget				
OPT	15.28	9870.8	416.835	+3.0
NO OPT	15.4	9718.1	404.8	

These results are showing the advantage of the metascheduling on the job-flow level. The next section describes the experiments on the application level.

4.2. Application level scheduling simulation results

The experiment results presented in Table 3 shows the advantage of the critical jobs method usage in a two-tier scheduling model compared to consecutive application-level scheduling.

Here k=0.75 means that each job is sent to be scheduled after 75% of the time allocated for the previous one: while the scheduling cost for a job is more or less the same, 1000 jobs are planned 25% faster.

Consider another experiment: while changing the length of the scheduling interval, we will estimate the proportion of successfully distributed jobs. The length of the scheduling interval is equal to $L = l * h, h = 1.0,..,2.6$, with step 0.2, where l is the length of the longest critical path of tasks in the job and h is a distribution interval magnification factor. There were carried 200 experiments for each h (bold points in Fig. 3).

Table 3. Two-tier model vs consecutive application-level scheduling

Parameter	Application-level scheduling	Two-tier model (k=0.75)
Jobs number	1000	1000
Execution time	531089 time units	399465 time units
Optimal schedules	687	703
Mean collision count	3.85	4.41
Mean load (forecast)	0.1843	0.1836
Mean load (fact)	0.1841	0.1830
Mean job cost	14.51 units	14.47 units

Analysis of the Fig. 3 shows that increasing the scheduling interval (relatively to the execution time of the longest critical path on the nodes with the highest performance) is accompanied by a significant increase in the number of successfully distributed jobs. The detailed study of this dependence can give a priori estimates of an individual job successful distribution probability.

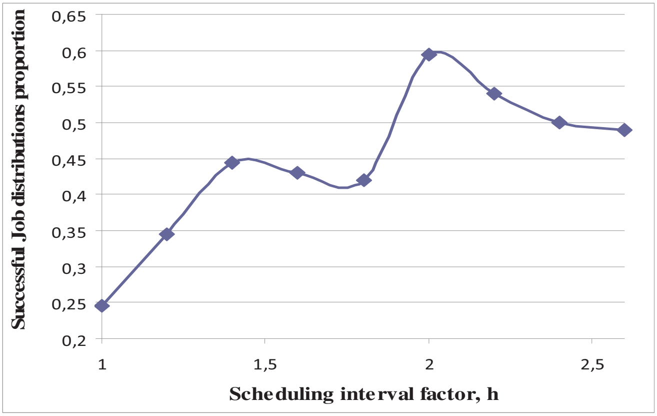
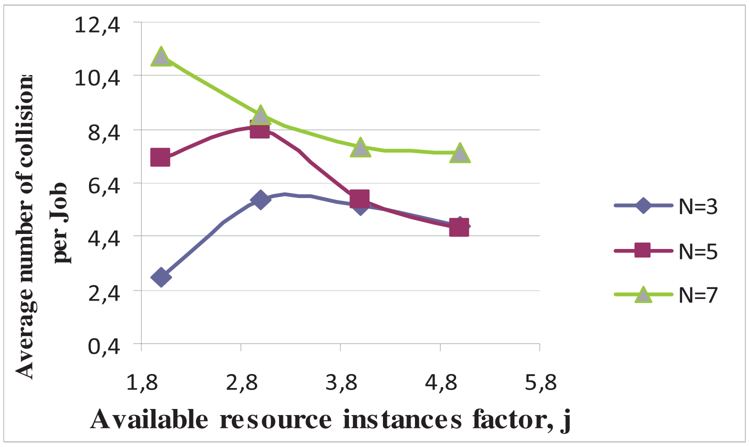
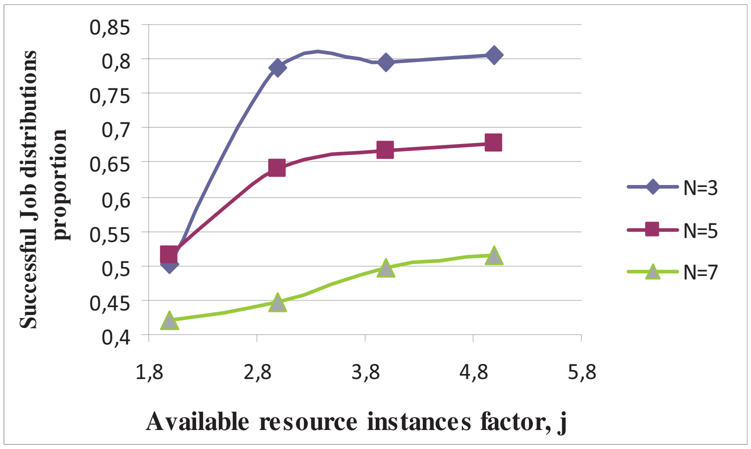


Fig. 3. Dependence of the proportion of the successful job distributions on the length of the distribution interval

In the next experiment we will consider the dependence of successful distributions number and the number of collisions per experiment on the level of resource instances availability. The experiments were performed in conditions of limited resources using the specific instances of the resources. The number of resources J in each experiment was determined as $J = j * N$, where j – factor (x-axis) and N – number of tiers in the graph. Fig. 4 shows results of the experiments with different j values and $N = 3,5,7$. The obtained dependencies (Fig. 4) suggest that the collisions number depends on the resources availability. The lower the number of resource instances and the greater the number of tiers in the graph – the more collisions occurred during the scheduling. At the same time the number of resource instances affects the successful distribution probability. With a value of $j > 4$ (that is, when the number of available resource instances is more than 4 times greater than the number of tiers in the graph) all cases provide the maximum value of successful distribution probability. These results are subject of future research of refined strategies on a job-flow level.



(a)



(b)

Fig. 4. Simulation results: (a) resource dependencies of collisions number; (b) successful job distribution proportion

5. Conclusions

In this work, we address the problem of independent job-flow scheduling in heterogeneous environment with non-dedicated resources.

Each job consists of a number of interrelated tasks with data dependencies. Using the combination of existing methods with a number of original algorithms the resulting schedules are computed. These schedules meet the defined deadlines and budget expectations, provide optimal load-balance for all the resources and follows VO strategies, thus, allowing to achieve unprecedented QoS and economic competitiveness for distributed systems such as Grid. The experiments which were conducted are showing the efficiency of methods developed for both job-flow and application level scheduling. The model proposed is showing the way these methods and advantages can be converged in one place making it possible to achieve the main goal. Currently there is no direct comparison with the existing systems made due to the fact, that today's systems do not perform optimization on both job-flow and application levels.

Future research will include the simulation of connected job-flow and application levels and experiments on real Grid-jobs in order to get finer view on advantages of the approach proposed.

Acknowledgements

This work was partially supported by the Council on Grants of the President of the Russian Federation for State Support of Leading Scientific Schools (SS-316.2012.9), the Russian Foundation for Basic Research (grant no. 12-07-00042), and by the Federal Target Program "Research and scientific-pedagogical cadres of innovative Russia" (state contracts nos. 16.740.11.0038 and 16.740.11.0516).

References

1. S.K. Garg, R. Buyya, H.J. Siegel, Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-off Management. Proc of ACSC 2009, Wellington, New Zealand (2009) 151-159.

2. G. Tesauro, J.L. Bredin, Strategic Sequential Bidding in Auctions Using Dynamic Programming. Proc. of the First International Joint Conference on Autonomous Agents and Multiagent Systems: part 2, ACM New York, NY, USA (2002) 591 – 598.

3. S.K. Garg, C.S. Yeo, A. Anandasivam, R. Buyya, Environment-conscious Scheduling of HPC Applications on Distributed Cloud-oriented Data Centers. Journal of Parallel and Distributed Computing. 71 (6) (2011) 732-749.

4. R. Buyya, D. Abramson, J. Giddy, Economic Models for Resource Management and Scheduling in Grid computing. J. of Concurrency and Computation: Practice and Experience. 14(5) (2002) 1507–1542.

5. C. Ernemann, V. Hamscher, R. Yahyapour, Economic Scheduling in Grid Computing. Proc. of the 8th Job Scheduling Strategies for Parallel Processing. Eds D.G. Feitelson, L. Rudolph, U. Schwiegelshohn. Heidelberg: Springer, LNCS. 2537 (2002) 128-152.

6. V. Voevodin, The Solution of Large Problems in Distributed Computational Media. Automation and Remote Control. Pleiades Publishing, Inc. 68 (5) (2007) 773-786.

7. K. Kurowski, J. Nabrzyski, A. Oleksiak et al., Multicriteria Aspects of Grid Resource Management. Grid resource management. State of the art and future trends. Eds J. Nabrzyski, J.M. Schopf and J. Weglarz. Kluwer Acad. Publ. (2003) 271–293.

8. D. Thain, T. Tannenbaum, M. Livny, Distributed Computing in Practice: the Condor Experience. J. of Concurrency and Computation: Practice and Experience 17 (2-4) (2004) 323 – 356.

9. V. Toporkov, Application-level and Job-flow Scheduling: an Approach for Achieving Quality of Service in Distributed Computing. Proc. of PaCT 2009, LNCS 5698. Berlin, Heidelberg (2009) 350 – 359.

10. V.V. Toporkov, Job and Application-level Scheduling in Distributed Computing. Ubiquitous Comput. Commun. J. 4 (2009) 559-570.

11. H. Dail, O. Sievert, F. Berman et al., Scheduling in the Grid Application Development Software project. Grid resource management. State of the art and future trends. Eds J. Nabrzyski, J.M. Schopf and J. Weglarz. Kluwer Acad. Publ. (2003) 73 – 98

12. Moab Adaptive Computing Suite, http://www.adaptivecomputing.com/products/moab-adaptive-computing-suite.php.

13. V. Toporkov, A. Toporkova, A. Bobchenkov, D. Yemelyanov, Resource Selection Algorithms for Economic Scheduling in Distributed Systems. Procedia Computer Science. Elsevier. 4 (2011) 2267-2276.

14. V. Toporkov, D. Yemelyanov, A. Toporkova, A. Bobchenkov, Resource Co-allocation Algorithms for Job Batch Scheduling in Dependable Distributed Computing. Dependable Computer Systems. Springer-Verlag, AICS. 97. Berlin, Heidelberg (2011) 243-256.

15. V. Toporkov, A. Bobchenkov, A. Toporkova, A. Tselishchev, D. Yemelyanov, Slot Selection and Co-allocation for Economic Scheduling in Distributed Computing. Proc. of the 11th Intern. Conf. on Parallel Computing Technologies. Springer-Verlag, LNCS. 6873. Berlin, Heidelberg (2011) 368–383.

16. V.V. Toporkov, A.S. Tselishchev, Safety Scheduling Strategies in Distributed Computing. Intern. J. of Critical Computer-Based Systems. 1(1/2/3) (2010) 41-58.

17. M. Cecchi, F. Capannini, A. Dorigo et al., The gLite Workload Management System. J. Phys.: Conf. Ser. 219 (6) (2010) 062039.

18. J. Yu, R. Buyya, K. Ramamohanarao, Workflow Scheduling Algorithms for Grid Computing. Metaheuristics for Scheduling in Distributed Computing Environments, Studies in Computational Intelligence. 146. Springer-Verlag. Berlin Heidelberg (2008) 173–214.

19. A. Tselishchev, V.V. Toporkov, Compound Job Scheduling and Job-flows Management in Distributed Computing. Proc. of the 54 Int. Colloquium. Ilmenau, Germany (2009) 21 – 26.

20. D. Jackson, Q. Snell, M. Clement, Core Algorithms of the Maui Scheduler, Springer, Heidelberg, LNCS 2221 (2001) 87-102.