# Resource Co-allocation Algorithms for Job Batch Scheduling in Dependable Distributed Computing

Victor Toporkov[1], Dmitry Yemelyanov[1], Anna Toporkova[2], and Alexander Bobchenkov[1]

[1] Moscow Power Engineering Institute (Technical University), ul. Krasnokazarmennaya 14, Moscow, 111250 Russia
e-mail: `ToporkovVV@mpei.ru`, `{groddenator,yemelyanov.dmitry}@gmail.com`

[2] Moscow State Institute of Electronics and Mathematics (Technical University), Bolshoy Trekhsvyatitelsky per. 1-3/12, Moscow, 109028 Russia
e-mail: `annastan@mail.ru`

**Abstract.** This work presents slot selection algorithms in economic models for independent job batch scheduling in distributed computing with non-dedicated resources. Existing approaches towards resource co-allocation and multiprocessor job scheduling in economic models of distributed computing are based on search of time-slots in resource occupancy schedules. The sought time-slots must match requirements of necessary span, computational resource properties, and cost. Usually such scheduling methods consider only one suited variant of time-slot set. This work discloses a scheduling scheme that features multi-variant search. Two algorithms of linear complexity for search of alternative variants are proposed and compared. Having several optional resource configurations for each job makes an opportunity to perform an optimization of execution of the whole batch of jobs and to increase overall efficiency of scheduling.

## 1 Introduction

Job control is among the most difficult problems in the enterprise of distributed computing in the case of non-dedicated resources that are shared with their owners. One must take into account the heterogeneity, changing composition, different owners of different nodes, and the scale of the computing environment. Economic models of scheduling are based on the concept of fair resource distribution between users and owners of computational nodes. They are effectively used in such spheres of distributed computing as Grid [1], cloud computing [2], and multiagent systems [3].

Two lasting trends can be distinguished among various approaches to the organization of computations in distributed environments [4-6]. One of them is based on the use of available resources when the role of mediator between users and computation nodes is played by special application agents called brokers [7, 8]. The other trend is closely related to the creation of virtual organizations (VO) [4]; it is mainly oriented to grid systems [4-6].

Both approaches have certain advantages and disadvantages. The resource management systems based on the first approach are well scalable and can be adapted to specific features of various applications. Resource brokers usually implement some economic policy in accordance with the application-level scheduling concept [7, 8]. However, the use of various optimization criteria of job execution by independent users (when jobs may compete with each other) can deteriorate such integral characteristics as total execution time of a batch of jobs and resource utilization. The creation of VO naturally restricts the scalability of job control systems. Nevertheless the use of certain rules for allocation and consumption of resources makes it possible to improve the efficiency of resource planning and allocation at the level of job flows [5]. The corresponding functions are implemented within a hierarchical structure consisting of a metascheduler and subordinate job schedulers [4-6] that are controlled by the metascheduler and in turn interact with resource managers (e.g., with batch job processing systems). The set of specific VO rules allows overall increase in the quality of service (QoS) for jobs and resource usage efficiency. It is worth noting that both approaches assume that applications are scheduled based on dynamically changing information about the global environment; both approaches make it possible to implement various resource management scenarios. Hence, we may speak not only of a scheduling algorithm but rather of a scheduling strategy, that is, of a combination of various methods of external and local scheduling, data allocation methods etc. [9, 10].

The model proposed in [11] is based on the concept of fair resource distribution between users and owners of computational nodes by means of economic mechanisms in VO. Existing approaches towards resource co-allocation and multiprocessor job scheduling in economic models of distributed computing are based on search of time-slots in resource occupancy schedules. The sought time-slots must match requirements of necessary span, computational resource properties, and cost [7, 8, 11]. There is the description of some approaches to forming of different deadline and budget constrained strategies of scheduling in [7]. Heuristic algorithms for slot selection based on user defined utility functions are introduced in [8]. Usually economic scheduling techniques consider only one suited variant of time-slot set.

In this work, a scheduling scheme with multi-variant slot search is proposed. Having several optional resource configurations for each job makes an opportunity to perform an optimization of execution of the whole batch of jobs and to increase overall efficiency of scheduling and QoS. We propose two algorithms for slot selection that feature linear complexity $O(m)$, where $m$ is the number of available time-slots.

Existing slot search algorithms, such as backfilling [12], do not support environments with inseparable resources, and, moreover, their execution time grows substantially with increase of the slot number. Assuming that every node has at least one local job scheduled, the backfill algorithm has quadratic complexity in the slot number. Although backfilling supports multiprocessor jobs and is able to find a rectangular window of concurrent slots, this can be done provided that all available computational nodes have equal performance (processor clock speed), and tasks of any job are homogeneous. We take a step further, so proposed algorithms deal with heterogeneous resources and jobs, and can form non-rectangular time-slot windows as a result.

This work is organized as follows. Section 2 introduces a main scheduling scheme. In section 3 two algorithms for search of alternative slot sets are considered. The example of slot search is presented in section 4. Simulation results for comparison of proposed algorithms are described in Section 5. Section 6 summarizes the work and describes further research topics.

## 2 Main Scheduling Scheme

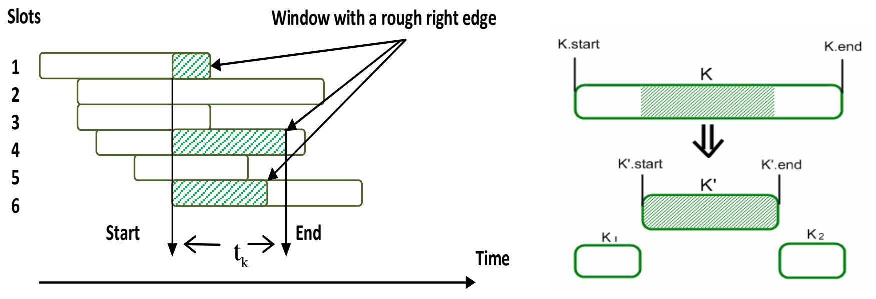
The job scheduling is finding a set of time slots. The resource requirements are arranged into a resource request containing the usage time $t$ and the characteristics of computational nodes (clock speed, RAM volume, disk space, operating system etc.).

Let $J = \{j_1, ..., j_n\}$ denote the batch consisting of $n$ jobs. A slot set fits a job $j_i, i = 1, ..., n$, if it meets the requirements of number and type of resources, cost $c_i$ and the job execution time $t_i$. We suppose that for each job $j_i$ in the current scheduling cycle there is at least one suitable set $s_i$. Otherwise, the scheduling of the job is postponed to the next iteration. Every slot set $s_i$ for the execution of the $i$-th job in a batch $J = \{j_1, ..., j_n\}$ is defined with a pair of parameters, the cost $c_i(s_i)$ and the time $t_i(s_i)$ of the resource usage, $c_i(s_i)$ denotes a total cost of slots in a set and $t_i(s_i)$ denotes the execution time of the $i$-th job.

During every cycle of the job batch scheduling two problems have to be solved.

1. Selecting alternative sets of slots (alternatives) that meet the requirements (resource, time, and cost).

2. Choosing a slot combination that would be the efficient or optimal in terms of the whole job batch execution in the current cycle of scheduling.



(a) (b)

**Fig. 1** Slot selection: an ordered list of available slots (a); slot subtraction (b)

To realize the scheduling scheme described above, first of all, we need to propose the algorithm of finding a set of alternative slot sets**.** Slots are arranged by start time in non-decreasing order (Fig. 1 (a)). In the case of homogeneous nodes, a set of slots for any job is represented with a rectangle window. In the case of CPUs with varying performance, that will be a window with a rough right edge, and the usage time is defined by the execution time $\mathrm{t_k}$ of the job part (task) that is using the slowest CPU.

This scheme works iteratively, during the iteration it consequentially searches for a single alternative for each job of the batch. In case of successful slot selection for the $i$-th job, the list of viewed slots for the $(i+1)$-th job is modified. All time spans that are involved in the $i$-th job alternative are excluded from the list of vacant slots (Fig. 1 (b)). The selection of slots for the $(i+1)$-th job is performed on the list modified with the method described above. Suppose, for example, that there is a slot K′ among the appropriate window slots. Then its start time equals to the start time of the window: K′.startTime = window.startTime and its end time equals to K′.end=K′.start $+\mathrm{t_k}$, where $\mathrm{t_k}$ is the evaluation of a task runtime on the CPU node, on which the slot is allocated. Slot K′ should be subtracted from the original list of available system slots. First, we need to find slot K – the slot, part of which is K′ and then cut K′ interval from K. So, in general, we need to remove slot K′ from the ordered slot list and insert two new slots $\mathrm{K_1}$ and $\mathrm{K_2}$. Their start, end times are defined as follows: $\mathrm{K_1}$.startTime = K.startTime, $\mathrm{K_1}$.endTime = K′.startTime, $\mathrm{K_2}$.startTime = K′.endTime, $\mathrm{K_2}$.endTime = K.endTime. Slots $\mathrm{K_1}$ and $\mathrm{K_2}$ have to be added to the slot list given that the list is sorted by non-decreasing start time order (see Fig. 1 (a)). Slot $\mathrm{K_1}$ will have the same position in the list as slot K, since they have the same start time. If slots $\mathrm{K_1}$ and $\mathrm{K_2}$ have a zero time span, it is not necessary to add them to the list.

After the last of the jobs is processed, the algorithm starts next iteration from the beginning of the batch and attempts to find other alternatives on the modified slot list. Alternatives found do not intersect in processor time, so every job could be assigned to some set of found slots without the revision of other jobs assignments. The search for alternatives ends when on the current list of slots the algorithm cannot find any suitable set of slots for any of the batch jobs.

An optimization technique for the second phase of this scheduling scheme was proposed in [11]. It is implemented by dynamic programming methods using multiple criteria in accordance with the VO economic policy.

We consider two types of criteria in the context of our model. These are the execution cost and time measures for the job batch $J$ using the suitable slot set $\bar{s}=(s_1,\ldots,s_n)$. The first criteria group includes the total cost of the job batch execution $C(\bar{s})=\sum_{i=1}^{n} c_i(s_i)$. The VO administration policy and, partially, users' interests are represented with the execution time criterion for all jobs of the batch

$T(\bar{s}) = \sum_{i=1}^{n} t_i(s_i)$. In order to forbid the monopolization of some resource usage by users, a limit $B^*$ is put on the budget of the VO that is the maximum value for a total usage cost of resources in the current scheduling cycle. The total slots occupancy time $T^*$ represents owners' urge towards the balance of global (external) and local (internal) job shares.

Let $g_i(s_i)$ be the particular function, which determines the efficiency of $s_i$ slot set usage for $i$-th job. In other words, $g_i(s_i) = c_i(s_i)$ or $g_i(s_i) = t_i(s_i)$. Let $f_i(Z_i)$ be the extreme value of the particular criterion using a slot set $s_i$ for execution of jobs $i, i+1, ..., n$, having $Z_i$ as a total occupancy time or the usage cost of slots $s_i, s_{i+1}, ..., s_n$ for jobs $j_i, j_{i+1}, ..., j_n$. Let us define an admissible time value or a slot occupancy cost as $z_i(s_i)$. Then $z_i(s_i) \le Z_i \le Z^*$, where $Z^*$ is the given limit. For example, if $z_i(s_i) = t_i(s_i)$, then $t_i(s_i) \le T_i \le T^*$, where $T_i$ is a total slots occupancy time for jobs $i, i+1, ..., n$ and $T^*$ is the constraint for values $T_i$, that is chosen with the consideration of balance between the global job flow (user-defined) and the local job flow (owner-defined). If, for example, $z_i(s_i) = c_i(s_i)$, then $c_i(s_i) \le C_i \le B^*$, where $C_i$ is a total cost of the resource usage for the tasks $i, i+1, ..., n$, and $B^*$ is the budget of the VO. In the scheme of backward run [13], $Z_1 = Z^*$, $i = 1$, $Z_i = Z_{i-1} - z_{i-1}(s_{i-1})$, having $1 < i \le n$.

The functional equation for obtaining a conditional (given $z_i(s_i)$) extremum of $f_i(z_i(s_i))$ for the backward run procedure can be written as follows:

$$f_i(Z_i) = \underset{s_i}{\operatorname{extr}} \{ g_i(s_i) + f_{i+1}(Z_i - z_i(s_i)) \},\ i = 1, ..., n,\ f_{n+1}(Z_{n+1}) \equiv 0. \quad (1)$$

If we consider the single-criterion optimization of the job batch execution, then every criterion $C(\bar{s})$ or $T(\bar{s})$ must be minimized with given constraints $T^*$ or $B^*$ for the interests of the particular party (a user, an owner and the VO administrator).

For example, a limit put on the total time of slot occupancy by tasks may be expressed as:

$$T^* = \sum_{i=1}^{n} \sum_{s_i} [t_i(s_i)/l_i], \quad (2)$$

where $l_i$ is the number of admissible slot sets for the $i$-th job; $[\cdot]$ means the nearest to $t_i(s_i)/l_i$ not greater integer.

The VO budget limit $B^*$ may be obtained as the maximal income for resource owners according to (1) with the given constraint $T^*$ defined by (2):

$$B^* = \max_{s_i}\{c_i(s_i) + f_{i+1}(T_i - t_i(s_i))\} . \tag{3}$$

In the general case of the model [7], it is necessary to use a vector of criteria, for example, $< C(\bar{s}), D(\bar{s}), T(\bar{s}), I(\bar{s}) >$ , where $D(\bar{s}) = B^* - C(\bar{s})$, and $I(\bar{s}) = T^* - T(\bar{s})$.

## 3 Slot Search Algorithms

Let us consider one of the resource requests associated with a job in a batch $J$. The resource requests specifies following: $N$ concurrent time-slots providing resource performance rate at least $P$ and maximal resource price not higher, than $C$, should be reserved for time span $t$. Here a slot search algorithm for a single job and resource charge per time unit is described.

It is an **A**lgorithm based on **L**ocal **P**rice of slots (ALP) with *a restriction to the cost of individual slots*. Source data include available slots list, and slots being sorted by start time in ascending order (see Fig. 1(a)). The search algorithm requires a sorted list to function and guarantees examination of every slot if this requirement is fulfilled.

**1**°. Sort the slots by start time in ascending order - see Fig. 1 (a).

**2**°. From the resulting slot list the next suited 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) $L(s_k) \geq t * P(s_k) / P$;

**c)** resource charge per time-unit $C(s_k) \leq C$.

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

**3**°. We add a time offset $d_k$ of current $k$-th slot in relation to $(k-1)$-th to the length of the window.

**4**°. Slots whose length has expired considering the offset $d_k$ are removed from the list. The expiration means that remaining slot length $L'(s_k)$, calculated like shown in **step 2°b**, is not enough assuming the $k$-th slot start is equal to the last added slot start: $L'(s_k) < (t - (T_{last} - T(s_k)))P(s_k) / P$, where $T(s_k)$ is the slot's start time, $T_{last}$ is the last added slot's start time.

**5**°. Go to **step 2**°, until the window has $N$ slots.

**6**°. **End** of the algorithm.

We can move only forward through the slot list. If we run out of slots before having accumulated $N$ slots, this means a failure to find the window for a job and its scheduling is postponed by the metascheduler until the next scheduling cycle. Otherwise, the window becomes an alternative slot set for the job. ALP is executed cyclically for every job in the batch $J = \{j_1, ..., j_n\}$. Having succeeded in the search for window for the $j_i$-th job, the slot list is modified with subtraction of formed window slots (see Fig. 1 (b)). Therefore slots of the already formed slot set are not considered in processing the next job in the batch.

In the economic model [11] a user's resource request contains the maximal resource price requirement, that is a price which a user agrees to pay for resource usage. But this approach narrows the search space and restrains the algorithm from construction of a window with more expensive slots. The difference of the next proposed algorithm is that we replace maximal price $C$ requirement by *a maximal budget of a job.*

It is an **A**lgorithm based on **M**aximal job **P**rice (AMP). The maximal budget is counted as $S = CtN$, where $t$ is a time span to reserve and $N$ is the necessary slot number. Then, as opposed to ALP, the search target is a window, formed by slots, whose total cost will not exceed the maximal budget $S$. In all other respects, AMP utilizes the same source data as ALP.

Let us denote additional variables as follows: $N_S$ – current number of slots in the window; $M_N$ – total cost of first $N$ slots.

Here we describe AMP approach for a single job.

**1**°. Find the earliest start window, formed by $N$ slots, using ALP excluding the condition **2°c** (see ALP description above).

**2**°. Sort window slots by their cost in ascending order.

Calculate total cost of first $N$ slots $M_N$.

If $M_N \leq S$, go to **4**°, so the resulting window is formed by first $N$ slots of the current window, others are returned to the source slot list. Otherwise, go to **3**°.

**3**°. Add the next suited slot to the list following to conditions **2°a** and **2°b** of ALP. Assign the new window start time and check expiration like in the **step 4**° of ALP.

If we have $N_S < N$, then repeat the current step. If $N_S \geq N$, then go to **step 2**°.

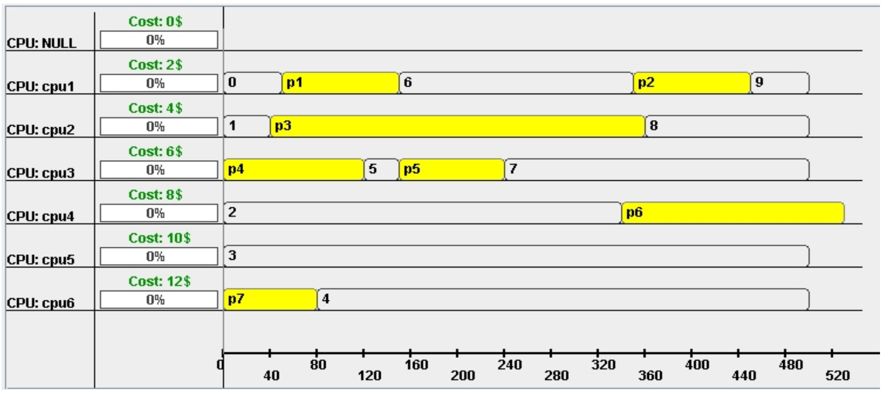
If we ran out of slots in the list, and $N_S < N$, then we have algorithm failure and no window is found for the job.

**4**°. **End** of the algorithm.

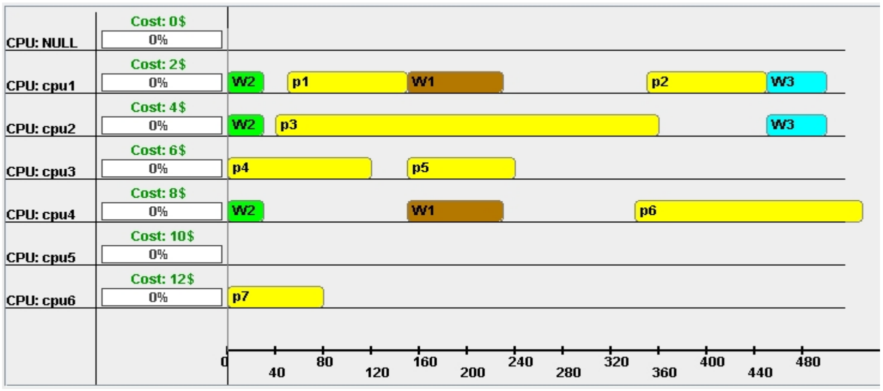
We can state three main features that distinguish the proposed algorithms. First, both *algorithms* consider resource performance rates. This allows forming time-slot windows with *uneven* right edge (we suppose that all concurrent slots for the job must start simultaneously). Second, both algorithms consider maximum price constraint which is imposed by a user. Third, both algorithms have linear complexity $O(m)$, where $m$ is the number of available time-slots.

## 4 AMP Search Example

In this example for the simplicity and ease of demonstration we consider the problem with a uniform set of resources, so the windows will have a rectangular shape without the rough right edge. Let us consider the following initial state of the distributed computing environment. In this case there are six processor nodes cpu1-cpu6 (Fig. 2 (a)). Each has its own unit cost (cost of it's usage per time unit), which is listed in the column to the right of the processor name. In addition there are seven local tasks p1-p7 already scheduled for the execution in the system under consideration. Available system slots are drawn as rectangles 0...9 - see Fig. 2 (a). Slots are sorted by non-decreasing time of start and the order number of each slot is indicated on its body. For the clarity, we consider the situation where the scheduling cycle includes the batch of only three jobs with the following resource requirements.



(a)



(b)

**Fig. 2** AMP: initial state of environment (a); alternatives found after the first iteration (b)

**Job 1** requirements: the number of required processor nodes: 2; runtime: 80; maximum total “window” cost per time: 10.

**Job 2** requirements: the number of required processor nodes: 3; runtime: 30; maximum total “window” cost per time: 30.

**Job 3** requirements: the number of required processor nodes: 2; runtime: 50; maximum total “window” cost per time: 6.

According to AMP alternatives search, first of all, we should form a list of available slots and find the earliest alternative (the first suitable window) for the first job of the batch. We assume that **Job 1** has the highest priority, while **Job 3** possesses the lowest priority. The alternative found for **Job 1** (see Fig. 2 (b)) has two rectangles on cpu1 and cpu4 resource lines on a time span [150, 230] and named W1.

The total cost per time of this window is 10. This is the earliest possible window satisfying the job’s resource request. Note that other possible windows with earlier start time are not fit the total cost constraint. Then we need to subtract this window from the list of available slots and find the earliest suitable set of slots for the second batch job on the modified list. Further, a similar operation for the third job is performed (see Fig. 2 (b)). Alternative windows found for each job of the batch are named W1, W2, and W3 respectively. The earliest suitable window for the second job (taking into account alternative W1 for the first job) consists of three slots on the cpu1, cpu2 and cpu4 processor lines with a total cost of 14 per time unit. The earliest possible alternative for the third job is W3 window on a time span of [450, 500]. Further, taking into account the previously found alternatives, the algorithm performs the searching of next alternative sets of slots according to the job priority. The algorithm works iteratively and makes an attempt to find alternative windows for each batch job at each iteration. Figure 3 illustrates the final chart of all alternatives found during search.

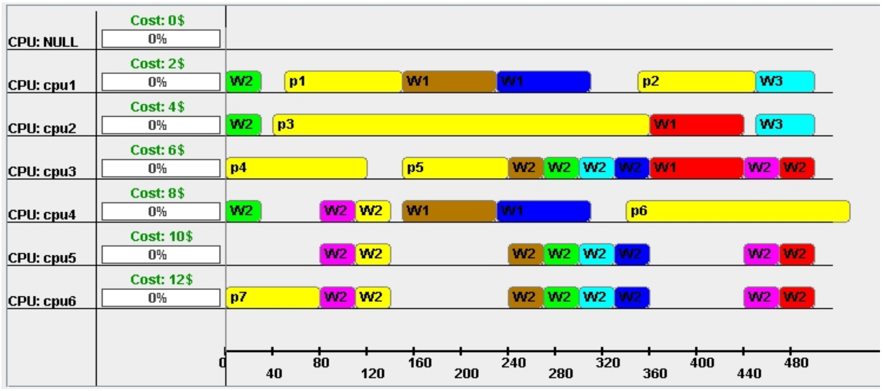


**Fig. 3** AMP: the final chart of all alternatives found

Note that in ALP approach the restriction to the cost of individual slots would be equal to 10 for **Job 2** (as it has a restriction of total cost equals to 30 for a window allocated on three processor nodes). So, processor cpu6 with a 12 usage cost value is not considered during the alternative search with ALP algorithm. However it is clear that in the presented AMP approach eight alternatives have been found. They use the slots allocated on cpu6 line, and thus fit into the limit of the window total cost.

## 5 Simulation Studies

The experiment consists in comparison of job batch scheduling results using different sets of suitable slots found with described above AMP and ALP approaches. The alternatives search is performed on the same set of available vacant system slots. During the single simulated scheduling cycle the generation of an ordered list of vacant slots and a job batch is performed. To perform a series of experiments we found it more convenient to generate an ordered list of available slots (see Fig. 1 (a)) with preassigned set of features instead of generating the whole distributed system model and obtain available slots from it. **SlotGenerator** and **JobGenerator** classes are used to form the ordered slot list and the job batch during the experiment series. Here is the description of the input parameters and values used during the simulation.

**SlotGenerator:**

- number of available system slots in the ordered list varies in [120, 150];
- length of the individual slot in [50, 300];
- computational nodes performance range is [1, 3];
- the probability that the nearby slots in the list have the same start time is 0.4;
- the time between neighbor slots in the list is in [0, 10];
- the price of the slot is randomly selected from [0.75p, 1.25p], where p= (1.7) to the (Node Performance).

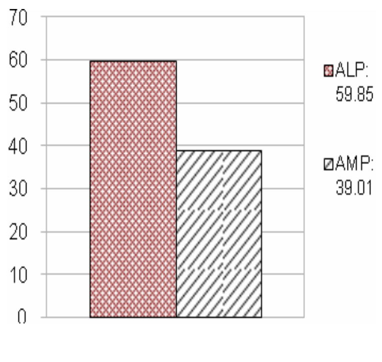
**JobGenerator:**

- number of jobs in the batch [3, 7];
- number of computational nodes to find is in [1, 6];
- length (representing the complexity) of the job [50, 150];
- the minimum required nodes performance [1, 2].

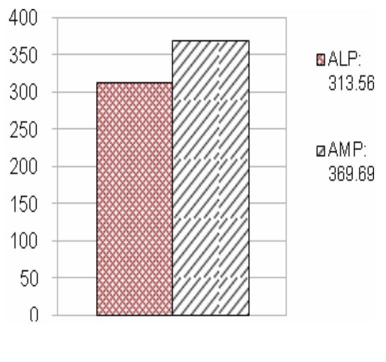
All job batch and slot list options are random variables that have a uniform distribution inside the identified intervals.

Let us consider the task of a slot allocation during the ***job batch total execution time minimization*** by the formula (1): $\min_{s_i} T(\bar{s})$ with the constraint $B^*$ in (3). We assume that in (1): $f_i(C_i) = \infty$ given $C_i = 0$.

The number of 25000 simulated scheduling cycles was carried out. Only those experiments were taken into account when all of the batch jobs had at least one suitable alternative of execution. When compared to the target optimization criterion, AMP algorithm exceeds ALP on 35%. Average batch job total execution time for alternatives found with ALP was 59.85, and for alternatives found with AMP: 39.01 (Fig. 4 (a)). It should be noted, that average cost of batch job execution for ALP method was 313.56, while using AMP algorithm average job execution cost was 369.69, that is 15% more – see Fig. 4 (b).



(a)



(b)

**Fig. 4** Job batch execution time minimization: job execution time (a); job execution cost (b)

Scheduling results comparison for the first 300 experiments can be viewed in Figure 5. It shows an observable gain of AMP method in every single experiment. The total number of alternatives found with ALP was 258079 or an average of 7.39 for a job. At the same time the modified approach (AMP) found 1160029 alternatives or an average of 34.28 for a single job.

According to the results of the experiment we can conclude that the use of AMP minimizes the total batch execution time though the cost of the execution increases. Relatively large number of alternatives found increases the variety of choosing the efficient slot combination [11] using the AMP algorithm.

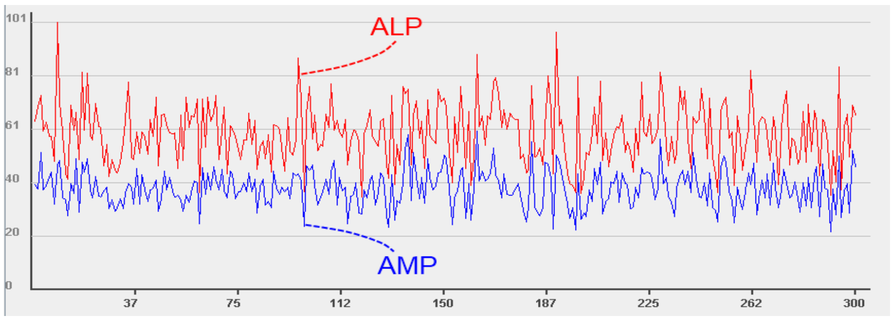


**Fig. 5** Execution time comparison for ALP and AMP in job batch execution time minimization

Now let us consider the task of slot allocation during the ***job batch total execution cost minimization*** by the formula (1): $\min_{s_i} C(\bar{s})$ with the constraint $T^*$ in (2). We assume that in (1): $f_i(T_i) = 0$ while $T_i = 0$.

The results of 8571 single experiments in which all batch jobs were successfully assigned to suitable set of resources using both slot search procedures were collected. Average batch job total execution cost for ALP algorithm was 313.09, and for alternatives found with AMP: 343.3.

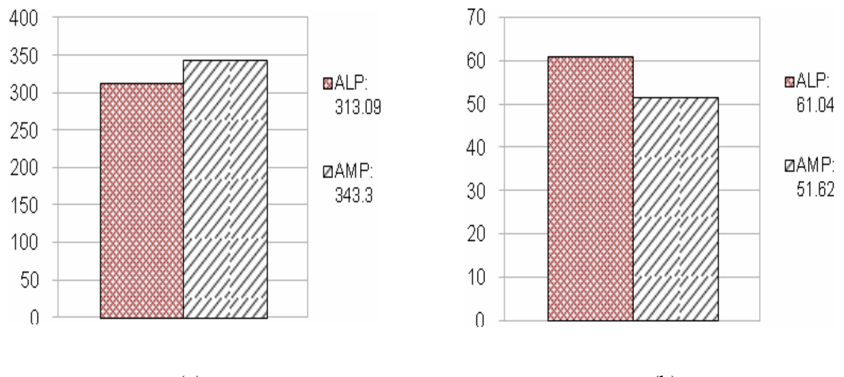


**Fig. 6** Job batch total execution cost minimization: job execution cost (a); job execution time (b)

It shows the advantage in the target criterion of only 9% for ALP approach over AMP (Fig. 6 (a)).

Average batch job total execution time for alternatives found with ALP was 61.04. Using AMP algorithm average job execution time was 51.62, that is 15% less than using ALP (Fig. 6 (b)).

Average number of slots processed in a single experiment was 135.11. This number coincides with the average number of slots for all 25000 experiments, which indicates the absence of decisive influence of available slots number to the number of successfully scheduled jobs. Average number of batch jobs in a single scheduling cycle was 4.18. This value is smaller than average over all 25000 experiments. With a large number of jobs in the batch ALP often was not able to find alternative sets of slots for certain jobs and an experiment was not taken into account. Average number of alternatives found with ALP is 253855 or an average of 7.28 per job. AMP algorithm was able to found a number of 115116 alternatives or an average of 34.23 per job. Recall that in previous set of experiments this numbers were 7.39 and 34.28 alternatives respectively.

According to the experimental results it can be argued that AMP allows to find at average more rapid alternatives and to perform jobs in a less time. But the total cost of job execution using AMP is relatively higher.

## 6 Conclusions and Future Work

In this work, we address the problem of independent batch jobs scheduling in heterogeneous environment with inseparable resources.

The feature of the approach is searching for a number of job alternative executions and consideration of economic policy in VO and financial user requirements on the stage of a single alternative search.

For this purpose ALP and AMP approaches for slot search and co-allocation were proposed and considered. When compared to the target optimization criteria during the total batch execution time minimization AMP exceeds ALP significantly. At the same time during the total execution cost minimization the gain of ALP method is negligible. It is worth noting, that on the same set of vacant slots AMP in comparison with ALP finds several time more execution alternatives. In our further work we will address the problem of slot selection for the whole job batch at once and not for each job consecutively.

The necessity of guaranteed job execution at the required QoS causes taking into account the distributed environment dynamics, namely, changes in the number of jobs for servicing, volumes of computations, possible failures of processor nodes, etc. [10, 14]. As a consequence, in the general case, a set of versions of scheduling, or a strategy, is required instead of a single version [10, 14].

In future we will refine resource co-allocation algorithms in order to integrate them with scalable co-scheduling strategies [14].

**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-7239.2010.9), the Russian Foundation for Basic Research (grant no. 09-01-00095), the Analytical Department Target Program "The higher school scientific potential development" (projects nos. 2.1.2/6718 and 2.1.2/13283), and by the Federal Target Program "Research and scientific-pedagogical cadres of innovative Russia" (State contracts nos. P2227 and 16.740.11.0038).

## References

[1] Garg, S.K., Buyya, R., Siegel, H.J.: Scheduling parallel applications on utility Grids: time and cost trade-off management. In: Proc. of ACSC 2009, pp. 151–159. Wellington, New Zealand (2009)

[2] Ailamaki, A., Dash, D., Kantere, V.: Economic aspects of cloud computing. Flash Informatique, Special HPC, 45–47 (October 27, 2009)

[3] Bredin, J., Kotz, D., Rus, D.: Economic markets as a means of open mobile-agent systems. In: Proc. of MAC 3, Seattle, USA, pp. 43–49 (1999)

[4] Kurowski, K., Nabrzyski, J., Oleksiak, A., et al.: Multicriteria aspects of Grid resource management. In: Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) Grid Resource Management. State of the art and future trends. Kluwer Academic Publishers, Boston (2003)

[5] Toporkov, V.: Application-level and job-flow scheduling: An approach for achieving quality of service in distributed computing. In: Malyshkin, V. (ed.) PaCT 2009. LNCS, vol. 5698, pp. 350–359. Springer, Heidelberg (2009)

[6] Toporkov,V.V.: Job and application-level scheduling in distributed computing. Ubiquitous Comput. Commun. J. 4, 559–570 (2009)

[7] Buyya, R., Abramson, D., Giddy, J.: Economic models for resource management and scheduling in grid computing. J. of Concurrency and Computation: Practice and Experience 5(14), 1507–1542 (2002)

[8] Ernemann, C., Hamscher, V., Yahyapour, R.: Economic scheduling in grid computing. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 128–152. Springer, Heidelberg (2002)

[9] Toporkov, V.V.: Decomposition schemes for synthesis of scheduling strategies in scalable systems. J. Comput. Syst. Sci. Int. 45, 77–88 (2006)

[10] Toporkov, V.V., Tselishchev, A.S.: Safety scheduling strategies in distributed computing. Int. J. Critical Computer-Based Syst. 1-3, 41–58 (2010)

[11] Toporkova, V.V., Toporkova, A., Tselishchev, A., Yemelyanov, D., Bobchenkov, A.: Economic models of scheduling in distributed systems. In: Walkowiak, T., Mazurkiewicz, J., Sugier, J., Zamojski, W. (eds.) Monographs of System Dependability. Dependability of Networks. Oficyna Wydawnicza Politechnki Wroclawskiej, Wroclaw (2010)

[12] Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. on Parallel and Distributed Systems 6(12), 529–543 (2001)

[13] Taha, H.: Operations research: an introduction. Macmillan, New York (1982)

[14] Toporkova, V.V., Toporkova, A., Tselishchev, A., Yemelyanov, D.: Scalable co-scheduling strategies in distributed computing. In: Proc. of the 2010 ACS/IEEE Int. Conf. on Computer Systems and Applications. IEEE CS Press, Los Alamitos (2010)