Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 2267–2276

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

Resource Selection Algorithms for Economic Scheduling in Distributed Systems

Victor Toporkov[a,*], Anna Toporkova[b], Alexander Bobchenkov[a], Dmitry Yemelyanov[a]

[a]*Computer Science Department, Moscow Power Engineering Institute (MPEI), ul. Krasnokazarmennaya, 14, Moscow, 111250, Russia*
[b]*Moscow State Institute of Electronics and Mathematics, Bolshoy Trekhsvyatitelsky per., 1-3/12, Moscow, 109028, Russia*

Abstract

In this paper, we present 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 paper discloses a scheduling scheme that features multi-variant search. Two algorithms of linear complexity for search of alternative variants are 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.

Keywords: scheduling; resource co-allocation; slot; resource request; job; batch; task

1. Introduction

Economic models for resource management and scheduling are based on the concept of fair resource distribution between users and owners of computational nodes. They are very effective in distributed computing, including Grid [1, 2], utility computing [3], cloud computing [4], and multiagent systems [5]. Pricing problem depending on the desired quality of service is a challenge in economic models of scheduling. There is a good overview of its solution in [6] along with the description of some approaches to forming of different deadline and budget constrained strategies of scheduling in distributed computation. In [7] heuristic algorithms for slot selection based on user defined utility functions are introduced. While implementing some economic policy, resource brokers usually optimize the performance of a specific application [1, 6, 7] in accordance with the application-level scheduling concept [8]. When establishing virtual organizations (VO), the optimization is performed for the job-flow scheduling [9, 10]. Corresponding functions are implemented by a hierarchical structure that consists of the metascheduler and subordinate resource managers or local batch-job management systems [8-10].

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

1877–0509 © 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of Prof. Mitsuhisa Sato and Prof. Satoshi Matsuoka
doi:10.1016/j.procs.2011.04.247

Within the framework of a model proposed in [2] there is an interaction between users launching their jobs and owners of computational resources. The interests of the said users and owners are often contradictory. Each independent user is interested in the earliest launch of his job with the lowest costs (for example, the resource usage fee) and the owners, on the contrary, try to make the highest income from their resources. In this work, economic mechanisms are applied for scheduling of a job batch in VO. It is supposed that resources are non-dedicated (inseparable), that is along with global flows of external user's jobs, owner's local job flows exist inside the CPU node domains. The metascheduler [8-10] implements the VO economic policy based on local CPU schedules. The schedules are sets of slots coming from local resource managers or schedulers in the node domains. A single slot is a time span that can be assigned to a task, which is a part of a multiprocessor job. We assume that scheduling of independent jobs runs iteratively (cycle by cycle) on dynamically updated local schedules [2]. The launch of a multiprocessor job requires a co-allocation of specified number of slots. The challenge is that slots associated with different CPU nodes may have arbitrary start and finish points that do not coincide. In its turn, processes of the parallel job must start synchronously. If the necessary number of slots with attributes matching the resource request is not accumulated then the job will not be launched.

We consider 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 [11, 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.

The paper 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. Experimental results are discussed in section 6. Section 7 summarizes the paper 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.).

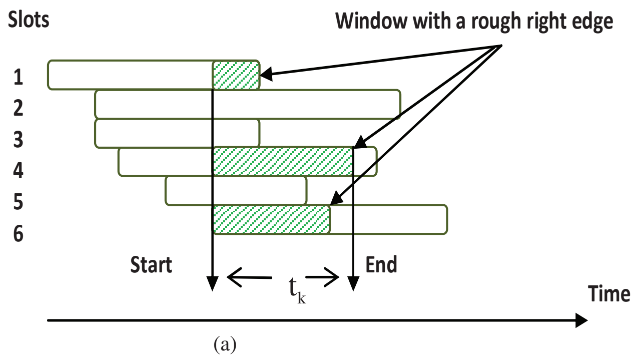
Let $J = \{j_1, ..., j_n\}$ denote the batch consisting of n jobs. A slot set fits a job $j_i, i = 1,...,n$, if it meets the requirements of number and type of resources, cost c_i and the job execution time t_i. We suppose that for each job j_i in the current scheduling cycle there is at least one suitable 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.

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 t_k of the job part (task) that is using the slowest CPU - see Fig . 1 (a).



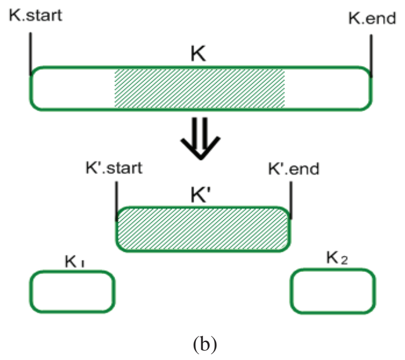


Fig. 1. Slot selection for heterogeneous resources: an ordered list of available slots (a); slot subtraction (b).

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 [2]. 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 $\bar{s}=(s_1,...,s_n)$. 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 [2] $Z_1 = Z^*$, $i = 1$, $0 \le Z_i \le Z^*$, 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:

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

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 [2].

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

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

where l_i is the number of admissible slot sets for the i-th job; $[\cdot]$ means 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 [2], 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 per time unit 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*. ALP likes a "conventional" approach for slot selection [7]. However we suppose its analysis in conjunction with the optimization method (Section 5). 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 $P(s_k) \ge P$; **b)** slot length (time span) is enough (depending on the actual performance of the slot's resource) $L(s_k) \ge tP(s_k)/P$; **c)** resource charge per time-unit $C(s_k) \le C$.

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 [2] 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 per time unit 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 $M_N \le S$, 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 $N_S \ge N$, 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 above 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: we move only forward through the list, and never return or reconsider previous assignments.

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.

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.

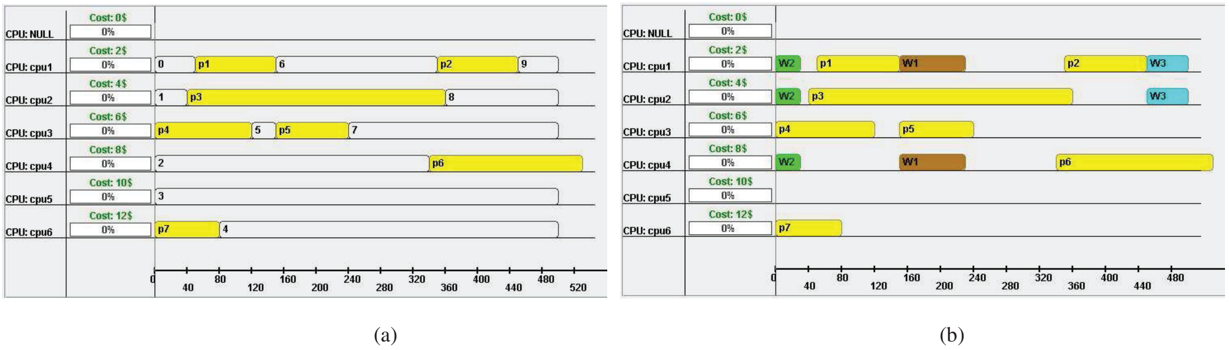


Fig. 2. AMP search example: initial state of distributed computing environment (a); alternatives found after the first iteration (b).

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.

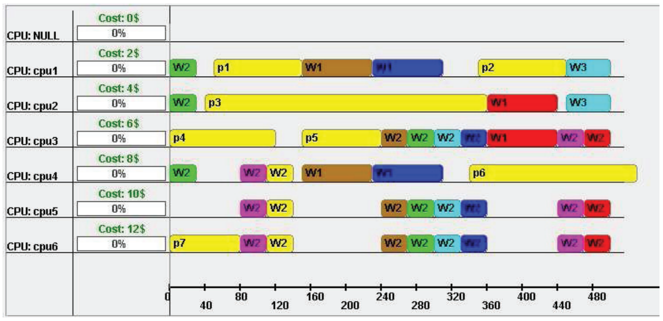


Fig. 3. The final chart of all alternatives found.

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], that is the environment is relatively homogeneous;
- the probability that the nearby slots in the list have the same start time is equal to 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).

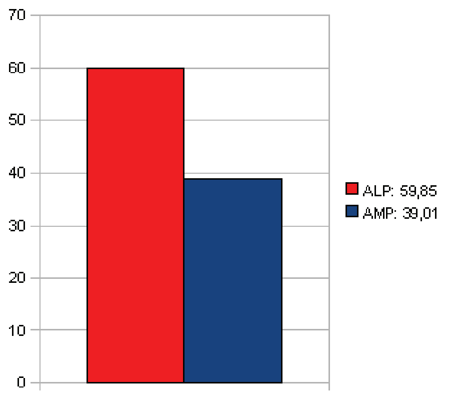
JobGenerator:

- 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].

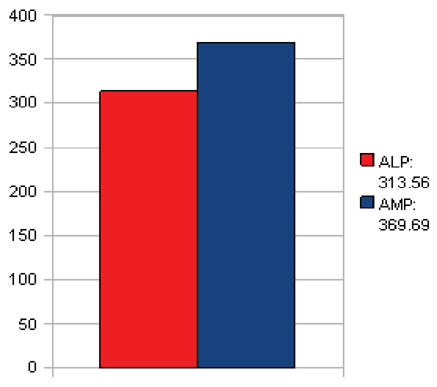
All job batch and slot list options are random variables that have a uniform distribution inside the identified intervals.

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). 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 modified approach (AMP) found 1160029 alternatives or an average of 34.28 for a single job.



(a)



(b)

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

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 [2] using the AMP algorithm.

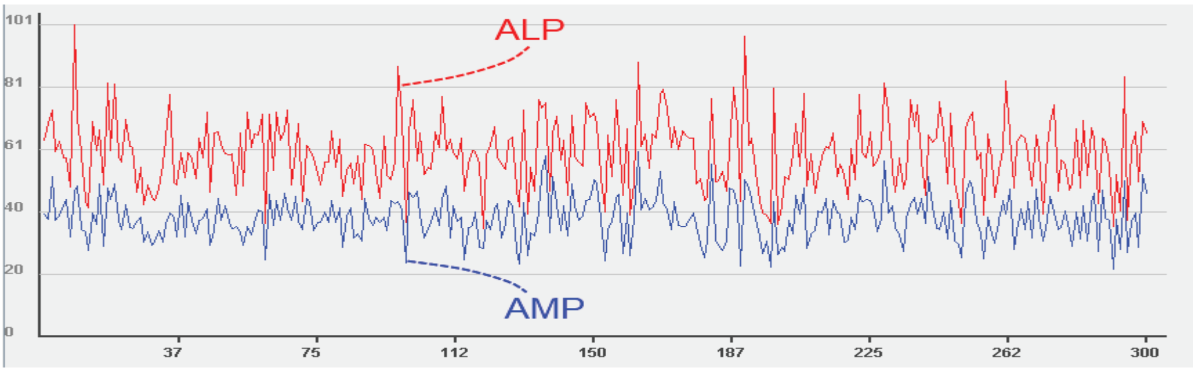
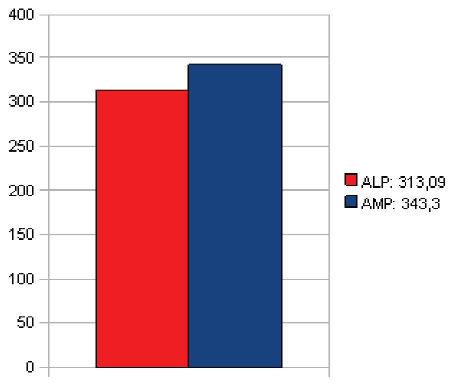


Fig. 5. Average job execution time comparison for ALP and AMP in job batch execution time minimization.

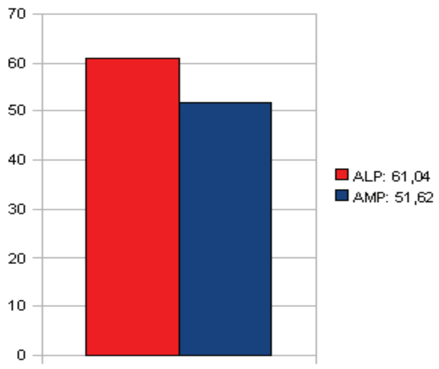
Now let us consider the task of slot allocation during the ***job batch total execution cost minimization*** by the formula (1): $\min_{s_i} C(\bar{s})$ with the constraint T^* in (2). We assume that in (1): $f_i(T_i)=0$ while $T_i=0$. The results of 8571 single experiments in which all batch jobs were 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. 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.



(a)



(b)

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

6. Discussion

Considering the results of the experiments it can be argued that the use of AMP approach on the stage of alternatives search gives clear advantage compared to the usage of "conventional" ALP. Advantages are mostly in large number of alternatives found and consequently in the flexibility of choosing an effective schedule of batch execution, as well as that AMP provides job total execution time less than ALP. AMP allows searching for alternatives among the relatively more expensive processor nodes with higher performance rate. Alternative sets of slots found with ALP are more homogeneous and do not differ much from each other by the values of total execution time and cost. Therefore job batch distributions obtained by optimizations based on various criteria [2] do not differ much from each other either. The following factors should explain the results. First, consider the peculiarities of calculating a slot usage total cost $C_t = CtN / P$, where C is a cost of slot usage per time unit, P is a relative performance rate of the processor node on which the slot is allocated, and t is a time span, required by the job in assumption that the job will be executed on the etalon nodes with $P = 1$. In proposed model, generally, the higher the cost C of slot the higher the performance P of node on which this slot is allocated. Hence, the job execution time t / P correspondingly less. So, the high slot cost per time unit is compensated by high performance of the CPU node, so it gets less time to perform the job and less time units to pay for.

Thus, in some cases the total execution cost may remain the same even with the more "expensive" slots. The value C / P is a measure of a slot price/quality ratio. By setting in resource request the maximum cost C of an individual slot and minimum performance rate P of a node the user specifies the minimum acceptable value of price/quality. The difference between ALP and AMP approaches lies in the fact that ALP searches for alternatives with suitable price/quality coefficient among the slots with usage cost no more than C. AMP performs the search among all the available slots (naturally, both algorithms still have the restriction on the minimum acceptable node performance). This explains why alternatives found with AMP have on average less total execution time. Second, it should be noted that during the search ALP considers available slots regardless of the entire "window". The ALP window consists of slots each of which has cost value no more than C. At the same time AMP is more flexible. If at some step a slot with cost on δ cheaper than C was added to the desired window, then AMP algorithm will consider to add slots with cost on the δ more expensive than C on the next steps. Naturally, in this case it will take into account the total cost restriction. That explains, why the average job execution cost is more when using the AMP algorithm, it seeks to use the entire budget to find the earliest suitable alternative. Another remark concerns the algorithm's work on the same set of slots. It can be argued that *any* window which could be found with ALP can also be found by AMP. However, there could be windows found with AMP algorithm which cannot be found with "conventional" ALP. It is enough to find a window that would contain at least one slot with the cost more than C. This observation once again explains the advantage of AMP approach by a number of alternatives found. The deficiency of AMP scheme is that total batch execution cost on average always higher than the execution cost of the same batch scheduled using ALP algorithm. It is a consequence of a specificity of determining the value of a budget limit and the stage of job batch scheduling [2]. However, it is possible to reduce the job batch total execution cost reducing the user budget limit for every alternative found during the search, which in this experiment was limited to $S = CtN$. This formula can be modified to $S = \rho CtN$, where ρ is a positive number less than one, e.g. 0.8. Variation of ρ allows to obtain flexible distribution schedules on different scheduling cycles, depending on the time of day, resource load level, etc. [2].

7. Conclusion and Future Work

In this paper, we address the problem of independent batch jobs scheduling in heterogeneous environment with inseparable resources. The scheduling of the job batch includes two phases. First of all, the independent sets of suitable slots (alternatives of execution) have to be found for every job of the batch. The second phase is selecting the effective combination of alternative slots. 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 compared. 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.

When compared to the target optimization criteria during the total batch execution time minimization AMP exceeds "conventional" 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 future work we will address the problem of slot selection for the whole job batch at once and not for each job consecutively. Therewith it is supposed to optimize the schedule "on the fly" and not to allocate a dedicated phase during each cycle for this optimization. We will research pricing mechanisms that will take into account supply-and-demand trends for computing resources in virtual organization. The necessity of guaranteed job execution at the required quality of service 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. [13]. As a consequence, in the general case, a set of versions of scheduling, or a strategy, is required instead of a single version [13, 14]. In our further work we will refine resource co-allocation algorithms in order to integrate them with scalable co-scheduling strategies.

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 no. 2.1.2/6718, 2.1.2/13283), and by the Federal Target Program "Research and scientific-pedagogical cadres of innovative Russia" (State contracts no. P2227, 16.740.11.0038).

References

1. S.K. Garg, R. Buyya, and H.J. Siegel, Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-off Management, Proceedings of the 32nd Australasian Computer Science Conference (ACSC 2009), Wellington, New Zealand (2009) 151-159.

2. V.V. Toporkov, A. Toporkova, A. Tselishchev, D. Yemelyanov, and A. Bobchenkov, Economic Models of Scheduling in Distributed Systems, In: T. Walkowiak, J. Mazurkiewicz, J. Sugier, and W. Zamojski (eds.), Monographs of System Dependability. Dependability of Networks. – Wroclaw: Oficyna Wydawnicza Politechnki Wroclawskiej 2 (2010) 143-154.

3. J. P. Degabriele and D. Pym, Economic Aspects of a Utility Computing Service, Trusted Systems Laboratory HP Laboratories Bristol HPL-2007-101, Technical Report (2007) 1-23.

4. A. Ailamaki, D. Dash, and V. Kantere, Economic Aspects of Cloud Computing, Flash Informatique, Special HPC (2009) 45-47.

5. J. Bredin, D. Kotz, and D. Rus, Economic Markets as a Means of Open Mobile-Agent Systems, Proceedings of the Workshop "Mobile Agents in the Context of Competition and Cooperation (MAC3)", 1999, 43-49.

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

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

8. K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz, Multicriteria Aspects of Grid Resource Management, In: Grid resource management. State of the art and future trends, J. Nabrzyski, J.M. Schopf, and J.Weglarz (eds.): Kluwer Academic Publishers, 2003, 271 – 293.

9. V. Toporkov, Application-level and Job-flow Scheduling: an Approach for Achieving Quality of Service in Distributed Computing, Proceedings of the 10th International Conference on Parallel Computing Technologies, Springer, Heidelberg, LNCS 5698 (2009) 350 – 359.

10. V.V. Toporkov, Job and Application-Level Scheduling in Distributed Computing, Ubiquitous Computing and Communication Journal. Special Issue on ICIT 2009 Conference (selected papers) - Applied Computing 3 (4) (2009) 559 – 570.

11. A.W. Mu'alem and D.G. Feitelson, Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling, IEEE Transactions on Parallel and Distributed Systems 6 (12) (2001) 529-543.

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

13. V.V. Toporkov and A. Tselishchev, Safety Scheduling Strategies in Distributed Computing, International Journal of Critical Computer-Based Systems, 1/2/3 (1) (2010) 41-58.

14. V.V. Toporkov, A. Toporkova, A. Tselishchev, and D. Yemelyanov, Scalable Co-Scheduling Strategies in Distributed Computing, Proceedings of the 2010 ACS/IEEE Int. Conference on Computer Systems and Applications, IEEE CS Press (2010) ISBN: 978-1-4244-7715-9.