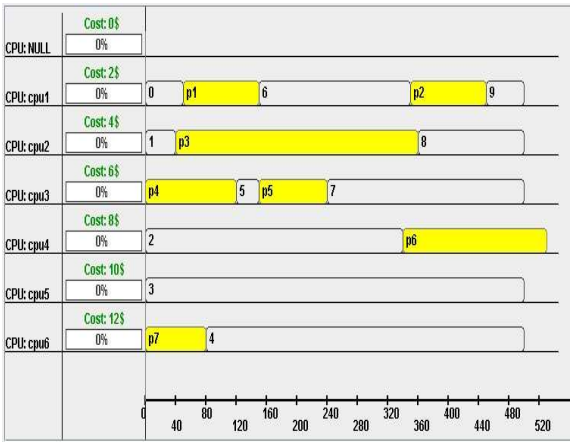

Figure 9: Initial state of distributed environment

Available system slots are drawn as rectangles 0...9 (see Fig. 9). 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

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 *Job1* has the highest priority, while *Job3* possesses the lowest priority.

The alternative found for *Job 1* (Fig. 10) 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. 10). 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].

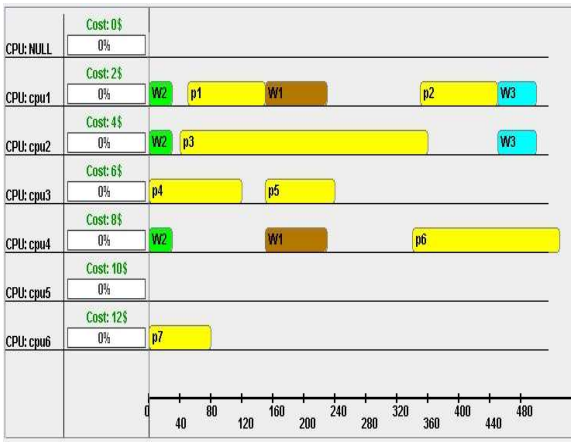

Figure 10: Alternatives found after the first iteration

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 (Fig. 11). Fig. 11 illustrates the final chart of all alternatives found during search. 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.

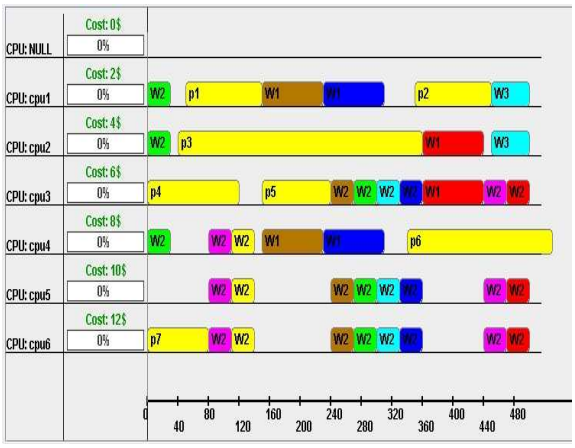

Figure 11: The final chart of all alternatives found

5 SIMULATION STUDIES

The experiment consists in comparison of job batch scheduling results using different sets of suitable slots founded 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.

The alternatives are found using algorithms ALP and AMP realized in **SlotProcessor** class with the main function

```
public slotProcessorResult
findAlternatives(ArrayList<ResourceRequest>
requests,
VOEnvironment environment, slotProcessorSettings
settings)
```

To perform a series of experiments we found it more convenient to generate an ordered list of available slots (see Fig. 8) with preassigned set of features instead of generating the whole distributed system model and obtaining available slots from it.

SlotProcessor class was used in order to carry out the experiment series. It realizes described ALP and AMP window search using the list of available slots in no decreasing start time order as an input parameter.

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 P = 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 execution time minimization (TM)*** by the technique proposed in [2]. 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.

AMP algorithm exceeds ALP by 35% with respect to the target optimization. An average batch job execution time for alternatives found with ALP was 59.85, and for alternatives found with AMP: 39.01 (Fig. 12).

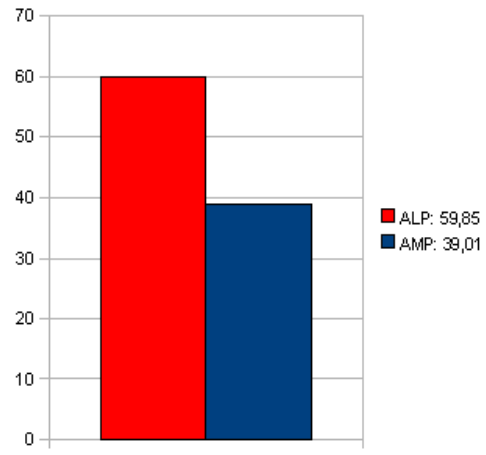

Figure 12: Total job execution time using ALP and AMP in TM

It should be noted, that an 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 (Fig. 13).

Scheduling results comparison for the first 300 experiments can be viewed in Fig. 14. It shows an observable gain of AMP method in every single experiment. The total number of alternatives found by 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 experiments 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.

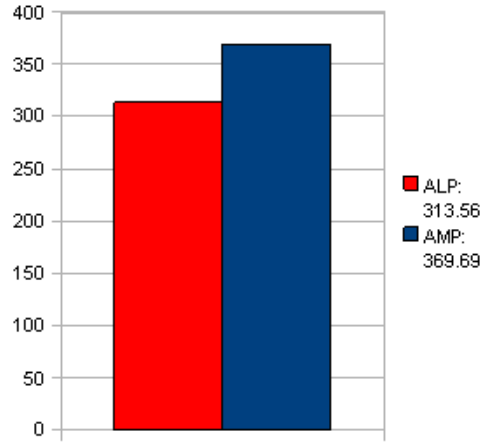

Figure 13: Total job execution cost using ALP and AMP in TM

Now let us consider the task of slot allocation during the ***job batch execution cost minimization (CM)*** [2].

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.

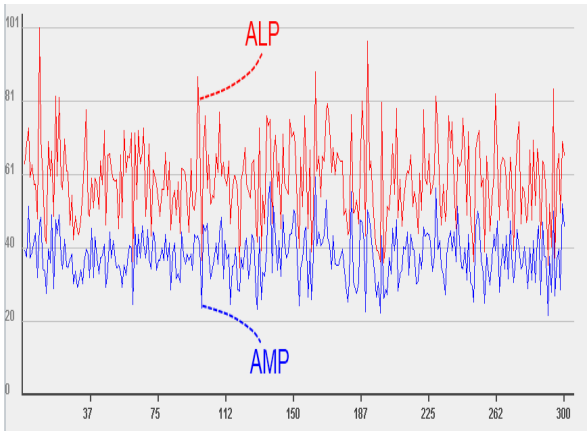

Figure 14: Average job execution time comparison for ALP and AMP in TM for the first 300 experiments

Average batch job execution cost for ALP algorithm was 313.09, and for alternatives found with AMP: 343.3. It shows the advantage in target criterion of only 9% for ALP approach over AMP (Fig. 15). 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. 16). 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 an alternative sets of slots for a certain jobs and 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.

6 DISCUSSION OF EXPERIMENTAL RESULTS

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 ALP.

Advantages are mostly in the large number of alternatives found and consequently in the flexibility of choosing an efficient schedule of batch execution, as well as that AMP provides the job batch execution time less than ALP.

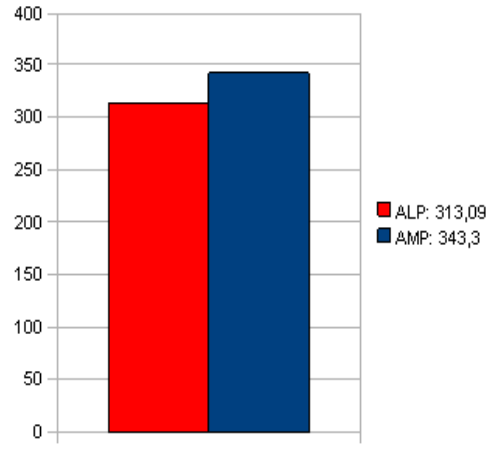

Figure 15: Total job execution cost using ALP and AMP in CM

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.

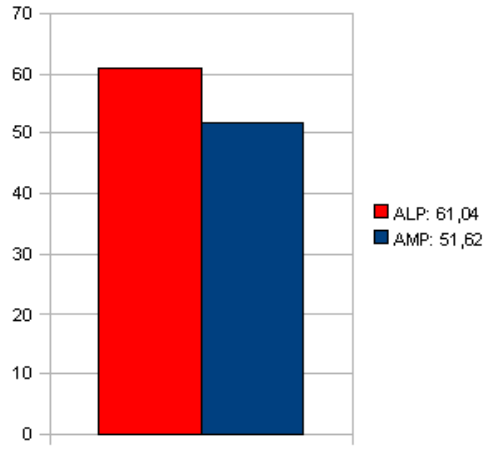

Figure 16: Total job execution time using ALP and AMP in CM

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 the resource request the maximum cost C of an individual slot and the 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 the 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 a conventional ALP. It is enough to find a window that would contain at least one slot with cost $C_S > C$. 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 $S = kCtN$, where k is a positive number less than one, e.g. 0.8. Variation of k allows to obtain flexible distribution schedules on different scheduling cycles, depending on the time of day, resource load level, etc. [2].

7 CONCLUSIONS AND FUTURE WORK

In this work, we address the problem of independent jobs batch scheduling in heterogeneous environment with non-dedicated 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 proposed and considered.

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 ALP significantly. At the same time during the 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 VO.

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.

ACKNOWLEDGMENT

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, 2.1.2/13283), and by the Federal Target Program "Research and scientific-pedagogical cadres of innovative Russia" (State contracts nos. P2227, 16.740.11.0038, 16.740.11.0516).

8 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, pp. 151-159 (2009).

[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 (Vol. 1-3). Dependability of Networks (Vol. 2). – Wroclaw: Oficyna Wydawnicza Politechnki Wroclawskiej, pp. 143-154 (2010).

[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, July 3, pp. 1-23 (2007).

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

[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)", pp. 43-49 (1999).

[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, Vol. 14, No. 5, pp. 1507 – 1542, (2002).

[7] C. Ernemann, V. Hamscher, and R. Yahyapour, "Economic Scheduling in Grid Computing", In: Proceedings of the 8th Job Scheduling Strategies for Parallel Processing, D.G. Feitelson, L. Rudolph, U. Schwiegelshohn (eds.), Springer, Heidelberg, LNCS, vol. 2537, 2002, pp. 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, pp. 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, Vol. 5698, pp. 350-359 (2009).

[10] V. V. Toporkov: Job and Application-Level Scheduling in Distributed Computing, Ubiquitous Computing and Communication (UbiCC) Journal. Special Issue on ICIT 2009 Conference (selected papers) - Applied Computing, Vol. 4, No. 3, pp. 559-570 (2009).

[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, Vol. 12, No. 6, pp. 529-543 (2001).

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

[13] V. V. Toporkov and A. Tselishchev: Safety Scheduling Strategies in Distributed Computing, International Journal of Critical Computer-Based Systems, Vol. 1. No. 1/2/3, pp. 41-58 (2010).

[14] V. V. Toporkov, A. Toporkova, A. Tselishchev, and D. Yemelyanov: Scalable Co-Scheduling Strategies in Distributed Computing, Proceedings of the 2010 ACS/IEEE International Conference on Computer Systems and Applications, Hammamet, Tunisia, May 16-19th, 2010. IEEE CS Press, ISBN: 978-1-4244-7715-9 (2010).