Optimization of Resources Selection for Jobs Scheduling in Heterogeneous Distributed Computing Environments*

Victor Toporkov[0000−0002−1484−2255] and Dmitry Yemelyanov[0000−0002−9359−8245]

National Research University "Moscow Power Engineering Institute",
ul. Krasnokazarmennaya, 14, Moscow, 111250, Russia,
{ToporkovVV, YemelyanovDM}@mpei.ru

Abstract. In this work, we introduce slot selection and co-allocation algorithms for parallel jobs in distributed computing with non-dedicated and heterogeneous resources (clusters, CPU nodes equipped with multi-core processors, networks etc.). A single slot is a time span that can be assigned to a task, which is a part of a parallel job. The job launch requires a co-allocation of a specified number of slots starting and finishing synchronously. The challenge is that slots associated with different heterogeneous resources of distributed computing environments may have arbitrary start and finish points, different pricing policies. Some existing algorithms assign a job to the first set of slots matching the resource request without any optimization (the first fit type), while other algorithms are based on an exhaustive search. In this paper, algorithms for effective slot selection are studied and compared with known approaches. The novelty of the proposed approach is in a general algorithm selecting a set of slots efficient according to the specified criterion.

Keywords: Distributed computing, economic scheduling, resource management, slot, job, allocation, optimization

1 Introduction

Modern high-performance distributed computing systems (HPCS), including Grid, cloud and hybrid infrastructures provide access to large amounts of resources [1, 2]. These resources are typically required to execute parallel jobs submitted by HPCS users and include computing nodes, data storages, network channels, software, etc. The actual requirements for resources amount and types needed to execute a job are defined in resource requests and specifications provided by users.

* This work was partially supported by the Council on Grants of the President of the Russian Federation for State Support of Young Scientists (YPhD-2297.2017.9), RFBR (grants 18-07-00456 and 18-07-00534) and by the Ministry on Education and Science of the Russian Federation (project no. 2.9606.2017/8.9).

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45

HPCS organization and support bring certain economical expenses: purchase and installation of machinery equipment, power supplies, user support, etc. As a rule, HPCS users and service providers interact in economic terms and the resources are provided for a certain payment. Thus, as total user job execution budget is usually limited, we elaborate an actual task to optimize suitable resources selection in accordance with a job specification and a restriction to a total resources cost.

Economic mechanisms are used to solve problems like resource management and scheduling of jobs in a transparent and efficient way in distributed environments such as cloud computing and utility Grid. In [3], we elaborate a hierarchical model of resource management system which is functioning within a VO. Resource management is implemented using a structure consisting of a metascheduler and subordinate job schedulers that interact with batch job processing systems. The significant and important feature for approach proposed in [3] as well as for well-known scheduling solutions for distributed environments such as Grids [1, 2, 4–6], is the fact that the scheduling strategy is formed on a basis of efficiency criteria. The metascheduler [3, 6] implements the economic policy of a VO based on local resource schedules. The schedules are defined as sets of slots coming from resource managers or schedulers in the resource domains, i.e. time intervals when individual nodes are available to perform a part of a parallel job. In order to implement such scheduling schemes and policies, first of all, one needs an algorithm for finding sets of simultaneously available slots required for each job execution. Further we shall call such set of simultaneously available slots with the same start and finish times as execution *window*.

In this paper we study algorithms for optimal or near-optimal resources selection by a given criterion with the restriction to a total cost. Additionally we consider solutions to overcome complications with different resources types, their heterogeneity, pre-known reservations and maintenance works.

2 Related Works

The scheduling problem in Grid is NP-hard due to its combinatorial nature and many heuristic-based solutions have been proposed. In [5] heuristic algorithms for slot selection, based on user-defined utility functions, are introduced. NWIRE system [5] performs a slot window allocation based on the user defined efficiency criterion under the maximum total execution cost constraint. However, the optimization occurs only on the stage of the best found offer selection. First fit slot selection algorithms (backtrack [7] and NorduGrid [8] approaches) assign any job to the first set of slots matching the resource request conditions, while other algorithms use an exhaustive search [2, 9, 10] and some of them are based on a linear integer programming (IP) [2, 9] or mixed-integer programming (MIP) model [10]. Moab scheduler [11] implements the backfilling algorithm and during a slot window search does not take into account any additive constraints such as the minimum required storage volume or the maximum allowed total allocation cost. Moreover, it does not support environments with non-dedicated resources.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45

Modern distributed and cloud computing simulators GridSim and CloudSim [12, 13] provide tools for jobs execution and co-allocation of simultaneously available computing resources. Base simulator distributions perform First Fit allocation algorithms without any specific optimization. CloudAuction extension [13] of CloudSim implements a double auction to distribute datacenters' resources between a job flow with a fair allocation policy. All these algorithms consider price constraints on individual nodes and not on a total window allocation cost. However, as we showed in [14], algorithms with a total cost constraint are able to perform the search among a wider set of resources and increase the overall scheduling efficiency.

GrAS [15] is a Grid job-flow management system built over Maui scheduler [11]. In order to co-allocate already partially utilized and reserved resources GrAS operates on a set of slots preliminary sorted by their start time. Resources co-allocation algorithm retrieves a set of simultaneously available slots (a window) with the same start and finish times even in heterogeneous environments. However the algorithm stops after finding the first suitable window and, thus, doesn't perform any optimization except for window start time minimization.

Algorithm [16] performs job's response and finish time minimization and doesn't take into account constraint on a total allocation budget. [17] performs window search on a list of slots sorted by their start time, implements algorithms for window shifting and finish time minimization, doesn't support other optimization criteria and the overall job execution cost constraint.

AEP algorithm [18] performs window search with constraint on a total resources allocation cost, implements optimization according to a number of criteria, but doesn't support a general case optimization. Besides AEP doesn't guarantee same finish time for the window slots in heterogeneous environments and, thus, has limited practical applicability.

In this paper, we propose algorithms for effective slot selection based on user defined criteria that feature linear complexity on the number of the available slots during the job batch scheduling cycle. The novelty of the proposed approach consists in allocating a set of simultaneously available slots. The paper is organized as follows. Section 3 introduces a general scheme for searching slot sets efficient by the specified criterion. Then several implementations are proposed and considered. Section 4 contains simulation results for comparison of proposed and known algorithms. Section 5 summarizes the paper and describes further research topics.

3 Resource Selection Algorithm

3.1 Problem Statement

We consider a set R of heterogeneous computing nodes with different performance p_i and price c_i characteristics. Each node has a local utilization schedule known in advance for a considered scheduling horizon time L. A node may be turned off or on by the provider, transfered to a maintenance state, reserved to

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45

perform computational jobs. Thus, it's convenient to represent all available resources as a set of slots. Each slot corresponds to one computing node on which it's allocated and may be characterized by its performance and price.

In order to execute a parallel job one needs to allocate the specified number of simultaneously idle nodes ensuring user requirements from the resource request. The resource request specifies number n of nodes required simultaneously, their minimum applicable performance p, job's computational volume V and a maximum available resources allocation budget C. The required window length is defined based on a slot with the minimum performance. For example, if a window consists of slots with performances $p \in \{p_i, p_j\}$ and $p_i < p_j$, then we need to allocate all the slots for a time $T = \frac{V}{p_i}$. In this way V really defines a computational volume for each single node subtask. Common start and finish times ensure the possibility of inter-node communications during the whole job execution. The total cost of a window allocation is then calculated as $C_W = \sum_{i=1}^{n} T * c_i$.

These parameters constitute a formal generalization for resource requests common among distributed computing systems and simulators.

Additionally we introduce criterion f as a user preference for the particular job execution during the scheduling horizon L. f can take a form of any additive function and vary from a simple window start time or cost minimization to a general independent parameter maximization with the restriction to a total resources allocation cost C. As an example, one may want to allocate suitable resources with the maximum possible total data storage available before the specified deadline.

3.2 General Window Search Procedure

For a general window search procedure for the problem statement presented in Section 3.1, we combined core ideas and solutions from algorithm AEP [18] and systems [15, 17]. Both related algorithms perform window search procedure based on a list of slots retrieved from a heterogeneous computing environment.

Following is the general square window search algorithm. It allocates a set of n simultaneously available slots with performance $p_i > p$, for a time, required to compute V instructions on each node, with a restriction C on a total allocation cost and performs optimization according to criterion f. It takes a list of available slots ordered by their non-decreasing start time as input.

1. Initializing variables for the best criterion value and corresponding best window: $f_{max} = 0$, $W_{max} = \{\}$.
2. From the slots available we select different groups by node performance p_i. For example, group P_k contains resources allocated on nodes with performance $p_i \geq P_k$. Thus, one slot may be included in several groups.
3. Next is a cycle for all retrieved groups P_i starting from the max performance P_{max}. All the sub-items represent a cycle body.
   (a) The resources reservation time required to compute V instructions on a node with performance P_i is $T_i = \frac{V}{P_i}$.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45

(b) Initializing variable for a window candidates list $S_W = \{\}$.
(c) Next is a cycle for all slots s_i in group P_i starting from the slot with the minimum start time. The slots of group P_i should be ordered by their non-decreasing start time. All the sub-items represent a cycle body.
   i. If slot s_i doesn't satisfy user requirements (hardware, software, etc.) then continue to the next slot (3c).
   ii. If slot length $l(s_i) < T_i$ then continue to the next slot (3c).
   iii. Set the new window start time $W_i.start = s_i.start$.
   iv. Add slot s_i to the current window slot list S_W.
   v. Next a cycle to check all slots s_j inside S_W.
      A. If there are no slots in S_W with performance $P(s_j) == P_i$ then continue to the next slot (3c), as current slots combination in S_W was already considered for previous group P_{i-1}.
      B. If $W_i.start + T_i > s_j.end$ then remove slot s_j from S_W as it can't consist in a window with the new start time $W_i.start$.
   vi. If S_W size is greater or equal to n, then allocate from S_W a window W_i (a subset of n slots with start time $W_i.start$ and length T_i) with a maximum criterion value f_i and a total cost $C_i < C$. If $f_i > f_{max}$ then reassign $f_{max} = f_i$ and $W_{max} = W_i$.
4. End of algorithm. At the output variable W_{max} contains the resulting window with the maximum criterion value f_{max}.

In this algorithm a list of slots-candidates S_W *moves* through the ordered list of all slots from each performance group P_i. During each iteration, when a new slot is added to the list (step 3(c)vi), any combination of n slots from S_W can form a suitable window if satisfy a restriction on the maximum allocation cost. In (3(c)vi) an optimal subset of n slots is allocated from S_W according to the criterion f with a restriction on the total cost . If this intermediate window W_i provides better criterion value compared to the currently best value ($f_i > f_{max}$) then we reassign variables W_{max} and f_{max} with new values. In this a way the presented algorithm is similar to the maximum value search in an array of f_i values.

3.3 Optimal Slot Subset Allocation

Let us discuss in more details the procedure which allocates an optimal (according to a criterion f) subset of n slots out of S_W list (algorithm step 3(c)vi).

For some particular criterion functions f a straightforward subset allocation solution may be offered. For example for a window finish time minimization it is reasonable to return at step 3(c)vi the first n *cheapest* slots of S_W provided that they satisfy the restriction on the total cost. These n slots (as any other n slots from S_W at the current step) will provide $W_i.finish = W_i.start + T_i$, so we need to set $f_i = -(W_i.start + T_i)$ to minimize the finish time. And at the end of the algorithm variable W_{max} will represent a window with the minimum possible finish time $W_{max}.finish = -f_{max}$.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45

The same logic applies for a number of other important criteria, including window start time, finish time and a total cost minimization.

However in a general case we should consider a subset allocation problem with some additive criterion: $Z = \sum_{i=1}^{n} c_z(s_i)$, where $c_z(s_i) = z_i$ is a target optimization characteristic value provided by a single slot s_i of W_i.

In this way we can state the following problem of an optimal n - size window subset allocation out of m slots stored in S_W:

$$Z = x_1 z_1 + x_2 z_2 + \ldots + x_m z_m, \tag{1}$$

with the following restrictions:

$$x_1 c_1 + x_2 c_2 + \ldots + x_m c_m \leq C$$

$$x_1 + x_2 + \ldots + x_m = n$$

$$x_i \in \{0, 1\}, i = 1, \ldots, m,$$

where z_i is a target characteristic value provided by slot s_i, c_i is total cost required to allocate slot s_i for a time T_i, x_i - is a decision variable determining whether to allocate slot s_i ($x_i = 1$) or not ($x_i = 0$) for a window W_i.

This problem relates to the class of integer linear programming problems, which imposes obvious limitations on the practical methods to solve it. However we used 0-1 knapsack problem as a base for our implementation. Indeed, the classical 0-1 knapsack problem with a total weight C and items-slots with weights c_i and values z_i have the same formal model (1) except for extra restriction on the number of items required: $x_1 + x_2 + \ldots + x_m = n$. To take this into account we implemented the following dynamic programming recurrent scheme:

$$f_i(C_j, n_k) = \max\{f_{i-1}(C_j, n_k), f_{i-1}(C_j - c_i, n_k - 1) + z_i\}, \tag{2}$$

$$n_k = 1, \ldots, n, i = 1, \ldots, m, C_j = 1, \ldots, C,$$

where $f_i(C_j, n_k)$ defines the maximum Z criterion value for n_k - size window allocated out of first i slots from S_W for a budget C_j. For the actual implementation we initialized $f_i(C_j, 0) = 0$, meaning $Z = 0$ when we have no items in the knapsack. Then we perform forward propagation and calculate $f_i(C_j, n_k)$ values for $n_k = 1, \ldots, n$. For example $f_i(C_j, 1)$ stands for $Z \to max$ problem when we can have only one item in the knapsack. Based on $f_i(C_j, 1)$ we can calculate $f_i(C_j, 2)$ using (2) and so on. So after the forward induction procedure (2) is finished the maximum value $Z_{max} = f_m(C, n)$. x_i values are then obtained by a backward induction procedure.

An estimated computational complexity of the presented recurrent scheme is $O(m * n * C)$, which is n times harder compared to the original knapsack problem ($O(m * C)$). However in practical job resources allocation cases this overhead doesn't look very large as we may assume that $n << m$ and $n << C$. On the other hand, this subset allocation procedure (2) may be called multiple times during the general square window search algorithm (step 3(c)vi).

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45

4 Simulation Study

4.1 Simulation Environment Setup

An experiment was prepared as follows using a custom distributed environment simulator [3, 18]. For our purpose, it implements a heterogeneous resource domain model: nodes have different usage costs and performance levels. A space-shared resources allocation policy simulates a local queuing system (like in GridSim or CloudSim [12]) and, thus, each node can process only one task at any given simulation time.

During the experiment series we performed a window search operation for a job requesting $n = 7$ nodes with performance level $p_i >= 1$, computational volume $V = 800$ and a maximum budget allowed is $C = 644$. The computing environment includes 100 heterogeneous computational nodes. Each node performance level is given as a uniformly distributed random value in the interval [2, 10]. So the required window length may vary from 400 to 80 time units. The scheduling interval length is 1200 time quanta which is enough to run the job on nodes with the minimum performance. The additional resources load (advanced reservations, maintenance windows) is distributed hyper-geometrically resulting in up to 30% utilization for each node.

Additionally an independent value $q_i \in [0; 10]$ is randomly generated for each computing node i to compare algorithms against $Q = \sum_{i=1}^{n} q_i$ window allocation criterion.

4.2 Algorithms Comparison

We implemented the following window search algorithms based on the general window search procedure introduced in Section 3.2.

1. *FirstFit* performs a square window allocation in accordance with a general scheme described in Section 3.2. Returns first suitable and affordable window found [15, 17].
2. *MinFinish*, *MinRuntime* and *MinCost* implements general scheme and returns windows with a minimum finish time, runtime (the difference between finish and start times) and execution cost correspondingly.
3. *MaxQ* implements a general square window search procedure with an optimal slots subset allocation (2) to return a window with maximum total Q value.
4. *MultipleBest* algorithm searches for multiple non-intersecting alternative windows using FirstFit algorithm. When all possible window allocations are retrieved the algorithm searches among them for alternatives with the minimum start time, finish time, runtime, cost and the maximum Q. In this way *MultipleBest* is similar to [5] approach.

Fig. 1 presents average window start time, runtime and finish time obtained by these algorithms based on 3000 independent simulation experiments. As expected, *FirstFit*, *MinFinish* and *MultipleBest* have the same minimum window

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45

finish time. Furthermore, they were able to start window at the beginning of the scheduling interval during each experiment($t_{start} = 0$). This is quite a probable event, since we are allocating 7 nodes out of 100 available, however partially utilized, nodes.

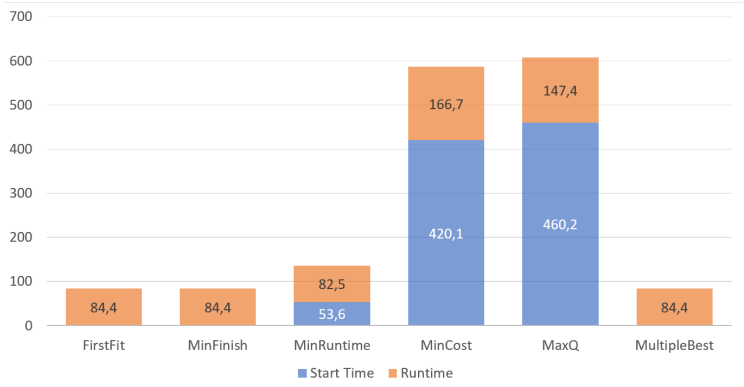

Fig. 1. Simulation results: average start time, runtime and finish time in computing environment with 100 nodes

Under such conditions *FirstFit* and *MinFinish* become practically the same algorithm: general window allocation scheme starts search among nodes with maximum performance. Thereby *FirstFit* combines minimum start time criterion with the maximum performance nodes. *MinRuntime* was able to slightly decrease runtime compared to *FirstFit* by using nodes with even higher performance, but starting a little later.

Windows allocated by *MinCost* and *MaxQ* are usually started closer to the middle of the scheduling interval. Late start time allowed these algorithms to perform a window search optimization among a wider variety of available nodes combinations. For example, average window allocation cost with the minimum value $C_W = 477$ is provided by *MinCost* (remember that we set $C = 644$ as a window allocation cost limit).*MinCost* advantage over *MultipleBest* approach is almost 17%. The advantage over other considered algorithms, not performing any cost optimization, reaches 24%.

Finally fig. 2 shows average $Q = \sum_{i=1}^{n} q_i$ value obtained during the simulation. Parameter q_i was generated randomly for each node i and is independent from node's cost, performance and slots start times. Thereby we use it to evaluate the general scheme (2) efficiency against optimization problem where no simple and accurate solution could possibly exist. Note that as q_i was generated randomly on a $[0; 10]$ interval and a single window should consist of 7 slots, we had the following practical limits specific for our experiment: $Q \in [0; 70]$.

As can be seen from fig. 2, *MaxQ* is indeed provided the maximum average value $Q = 61.8$, which is quite close to the practical maximum, especially com-

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45

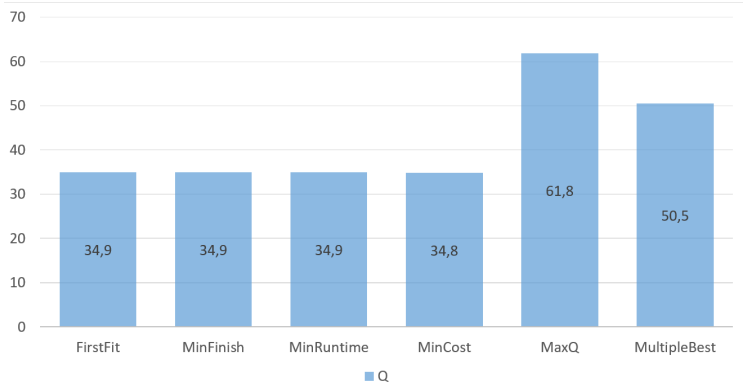

Fig. 2. Simulation results: average window Q value

pared to other algorithms. *MaxQ* advantage over *MultipleBest* is 18%. Other algorithms provided *average Q* value exactly in the middle of $[0; 70]$ interval and *MaxQ* advantage over them is almost 44%.

5 Conclusion and Future Work

In this work, we address the problem of slot selection and co-allocation for parallel jobs in distributed computing with non-dedicated resources. For this purpose a general *square* window allocation algorithm was proposed and considered. A special slots subset allocation procedure is implemented to support a general case optimization problem.

Simulation study proved algorithms' optimization efficiency according to their target criteria. A general case implementation showed 44% advantage over First Fit algorithms and 18% over a simplified *MultipleBest* optimization heuristic. As a drawback, the general case algorithm has a high computational complexity compared to *FirstFit*.

In our further work, we will refine resource co-allocation algorithms in order to decrease their computational complexity. Another research direction will be focused on a practical resources allocation tasks implementation based on the proposed general case approach.

References

1. Lee, Y.C., Wang C., Zomaya, A.Y., Zhou, B.B.: Profit-driven Scheduling for Cloud Services with Data Access Awareness. J. of Parallel and Distributed Computing 72 (4), 591-602 (2012)
2. Garg, S.K., Konugurthi, P., Buyya, R.: A Linear Programming-driven Genetic Algorithm for Meta-scheduling on Utility Grids. Int. J. of Parallel, Emergent and Distributed Systems 26, 493-517 (2011)

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45

3. Toporkov, V., Tselishchev, A., Yemelyanov, D., Bobchenkov, A.: Composite Scheduling Strategies in Distributed Computing with Non-dedicated Resources. Procedia Computer Science. Elsevier, vol. 9, pp. 176-185 (2012)
4. 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)
5. 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)
6. Kurowski, K., Nabrzyski, J., Oleksiak, A., Weglarz, J.: Multicriteria Aspects of Grid Re-source Management. In: Nabrzyski, J., Schopf, J.M., Weglarz J. (eds.) Grid resource management. State of the art and future trends. Kluwer Academic Publishers, pp. 271-293 (2003)
7. Aida, K., Casanova, H.: Scheduling Mixed-parallel Applications with Advance Reservations. In: 17th IEEE Int. Symposium on HPDC, pp. 65-74. IEEE CS Press, New York (2008)
8. Elmroth, E., Tordsson J.: A Standards-based Grid Resource Brokering Service Supporting Advance Reservations, Coallocation and Cross-Grid Interoperability. J. of Concurrency and Computation: Practice and Experience 25(18), 2298-2335 (2009)
9. Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y.: An Advance Reservation-based Co-allocation Algorithm for Distributed Computers and Network Bandwidth on QoS-guaranteed Grids. In: Frachtenberg E., Schwiegelshohn U. (eds.) JSSPP 2010. LNCS, vol. 6253, pp. 16-34. Springer, Heidelberg (2010)
10. Blanco, H., Guirado, F., Lrida, J.L., Albornoz, V.M.: MIP Model Scheduling for Multi-clusters. In: Euro-Par 2012. LNCS, vol. 7640, pp. 196-206. Springer, Heidelberg (2013)
11. Moab Adaptive Computing Suite, http://www.adaptivecomputing.com/
12. Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., and Buyya,R.: CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. J. Software: Practice and Experience, 41(1),23-50 (2011)
13. Samimi, P., Teimouri, Y., Mukhtar M.: A combinatorial double auction resource allocation model in cloud computing. J. Information Sciences, 357(C),201-216 (2016)
14. Toporkov, V., Toporkova, A., Bobchenkov, A., Yemelyanov, D.: Resource Selection Algorithms for Economic Scheduling in Distributed Systems. In: Proc. International Conference on Computational Science, ICCS 2011, June 1-3, 2011, Singapore, Procedia Computer Science. Elsevier, vol. 4. pp. 2267-2276 (2011)
15. Kovalenko, V.N., Kovalenko, E.I., Koryagin, D.A., et. al., Parallel Job Management in the Grid with Non-Dedicated Resources, Preprint of Keldysh Inst. of Appl. Math., Russ. Acad. Sci., Moscow, no. 63, 2007.
16. Makhlouf, S., Yagoubi, B.: Resources Co-allocation Strategies in Grid Computing. In: CIIA, vol. 825, CEUR Workshop Proceedings, 2011.
17. Netto, M. A. S., Buyya, R.: A Flexible Resource Co-Allocation Model based on Advance Reservations with Rescheduling Support. In: Technical Report, GRIDS-TR-2007-17, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, October 9, 2007.
18. Toporkov, V., Toporkova, A., Tselishchev, A., and Yemelyanov, D.: Slot Selection Algorithms in Distributed Computing. J. of Supercomputing, 69 (1), 53-60 (2014)

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_45