Allocation of Distributed Resources with Group Dependencies and Availability Uncertainties

Victor Toporkov [0000–0002–1484–2255], Dmitry Yemelyanov [0000–0002–9359–8245]

and Alexey Tselishchev

National Research University "MPEI", Russia
ToporkovVV@mpei.ru, YemelyanovDM@mpei.ru,
Alexey.Tselishchev@gmail.com

Abstract. In this work, we introduce and study a set of tree-based algorithms for resources allocation considering group dependencies between their parameters. Real world distributed and high-performance computing systems often operate under conditions of the resources availability uncertainty caused by uncertainties of jobs execution, inaccuracies in runtime predictions and other global and local utilization events. In this way we can observe an availability over time function for each resource and use it as a scheduling parameter. As a single parallel job usually occupies a set of resources, they shape groups with common probabilities of usage and release events. The novelty of the proposed approach is an efficient algorithm considering groupings of resources by the common availability probability for the resources' co-allocation. The proposed algorithm combines dynamic programming and greedy methods for the probability-based multiplicative knapsack problem with a tree-based branch and bounds approach. Simulation results and analysis are provided to compare different approaches, including greedy and brute force solution.

Keywords: Distributed Computing, Resource, Uncertainty, Availability, Probability, Job, Group, Knapsack, Branch and Bounds.

1 Introduction and Related Works

High-performance distributed computing systems, such as Grids, cloud, and hybrid infrastructures, provide access to substantial amounts of resources. These resources are typically required to execute parallel jobs submitted by 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 [1-5]. Distributed computing systems organization and support bring certain economical expenses: purchase and installation of machinery equipment, power supplies, user support, etc. As a rule, users and service providers interact in economic terms and the resources are provided for a certain payment. Economic models [3-5] are used to efficiently solve resource management and job-flow scheduling problems in distributed environments such as cloud computing and utility

Grids. Majority of scheduling solutions for distributed environments implement scheduling strategies on a basis of efficiency criteria [1–5].

Traditional models consider scheduling problem in a deterministic way. Such an approach is sometimes justified by the strict market rules for resources acquisition and utilization during the purchased period of time. Commercial Grids and cloud service providers usually own full control over the resources and may reliably consider their local schedules for some scheduling horizon time [1, 3]. Besides, market-based interactions and QoS constraints compliance require deterministic model for the resources utilization profile. Thus, it is convenient to represent available resources as a set of slots: time intervals when particular nodes are idle and may be used for user jobs execution [4-8]. However general distributed computing systems with non-dedicated resources usually cannot rely on deterministic utilization schedules and instead make predictions based on the utilization predictions and probabilities [9-12]. The probabilities of the resources' availability and utilization at any given time may originate from jobs execution and completion time uncertainties, local activities of the resource provider, maintenance, or numerous failure events. Particular utilization characteristics and patterns usually strongly depend on the resource types. However, according to [9] about 20% of Grid computational nodes exhibit truly random availability intervals.

The scheduling problem in Grid is *NP*-hard due to its combinatorial nature and many heuristic solutions have been proposed. When scheduling under uncertainties, proactive and reactive approaches are usually distinguished [12]. Proactive algorithms concentrate on the resources' utilization predictions and heuristic-based advanced resources allocations and reservations. Reactive algorithms analyze current state of the computing environment and make decisions for jobs migration and rescheduling. Both types of algorithms may be used in a single system to achieve even greater resource usage efficiency. The resources availability predictions for the considered scheduling interval may be obtained based on the historical data processing, linear regression models or with help of expert and machine learning systems [9-11]. In [10], a set of availability states is defined to model resource behavior and probabilities state transitions. On the other hand, sometimes it is possible to identify distributions of resources utilization and availability intervals [9]. Economic scheduling models are implemented in modern distributed and cloud computing simulators GridSim and CloudSim [13]. They provide reliable tools for resources co-allocation but consider price constraints on individual nodes and not on a total window allocation cost. However, as we showed in [6], algorithms with a total cost constraint can perform the search among a wider set of resources and increase the overall scheduling efficiency. Algorithms [14-16] implement knapsack- based slot selection optimization according to a probability-based criterion with a total job execution cost constraint.

This paper extends scheduling algorithms and model presented in [14-16]. We propose proactive algorithms for resources selection and co-allocation computing environments with non-dedicated resources and corresponding availability uncertainties. The uncertainties are modeled as resources availability events and probabilities: a natural way of machine learning and statistical predictions representation [16]. Common resources' allocation and release times are modeled with interdependent resource groups.

The novelty of the proposed approach consists in a dynamic programming scheme performing resources selection with a total availability criterion maximization. The paper is organized as follows. Section 2 presents availability-based scheduling problem and several greedy, knapsack and branch and bounds-based approaches for its solution. Section 3 contains an experiment setup and simulation results obtained for the considered algorithms. Section 4 summarizes the paper and highlights further research topics.

2 Resource Selection Algorithm

2.1 Probabilistic Model for Resource Utilization

In our model we consider a set R of heterogeneous computing nodes with price c_i characteristics under utilization uncertainties. The probabilities (predictions) $p_i(t)$ of the resources' availability and utilization for the whole scheduling interval L are provided as input data. We model a resource utilization schedule as an ordered list of utilization events, such as resource's *allocation, occupation (execution)* and *release* events. An individual job execution on a single resource is modeled as a sequence of *allocation*, *occupation* (actual execution) and *release* events (see Fig. 1). Additionally, global resources utilization uncertainties, such as maintenance works or network failures, are modeled as a continuous *occupation* event with $P_o \ll 1$ during the whole considered scheduling interval.

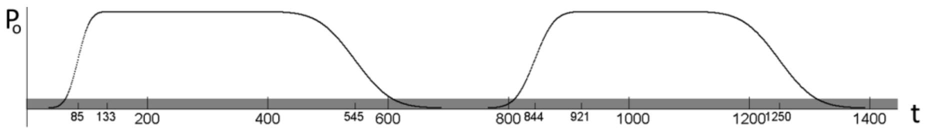

Fig. 1. Example of a single resource occupation probability schedule.

Fig. 1 shows an example of a single resource occupation probability P_o schedule. With two jobs already assigned to the resource, there are two resources allocation events (with expected times of allocation at 85- and 844-time units), two resources occupation events (starting at 133- and 921-time units) and two resources release events (expected release times are 545- and 1250-time units respectively). Gray translucent bar at the bottom of the Fig. 1 represents a sum of global utilization events with a total resource occupation probability $P_o = 0.06$. During the whole *execution* interval, the resource's occupation (utilization) probability is assumed as $P_o = 1$. Utilization probability for *allocation* events is modeled by random variable with a normal distribution, and for *release* events - with a *lognormal distribution* to consider the long tails [15]. Expected allocation and release times are derived from the job's replication and execution time estimations. Corresponding standard deviations depend on the job's features and may be predicted based on user estimations or historical data [9-11,15]. Hence, in Fig. 1 the resource occupation probability at expected times of allocation and release events are: $P_o(85) = P_o(545) = P_o(844) = P_o(1250) = 0.5$.

However, to execute a job, a resource should be allocated for a specified time period T. Based on the model above, we propose the following procedure to calculate a total availability probability P_a of a resource r during time interval T. P_a describes probability, that the resource r will be fully available and will not be interrupted during T.

1. Retrieve a set of independent utilization events e_i active for the resource r during the time interval T. When a subset of dependent events is active during the interval, then only a single event providing the maximum occupation probability P_o is retrieved. For example, from the *allocation-occupation (execution)-release* events chain only the *execution* event is retrieved with $P_o = 1$.
2. For each independent event e_i a maximum occupation probability during the interval l is calculated: $P_o^{max}(e_i) = \max_{t \in T} P_o(e_i, t)$. Corresponding partial availability probability $P_a(e_i)$ is calculated for each event e_i as a probability that the resource will not be occupied by the event during the interval T: $P_a(e_i) = 1 - P_o^{max}(e_i)$.
3. The resource will be available during the whole-time interval T only in case it will not be occupied by any of the active utilization events. Thus, the total availability probability for the resource r is a product of all partial availability probabilities calculated for independent events e_i:

$$P_a^r = \prod_i P_a(e_i)\ . \tag{1}$$

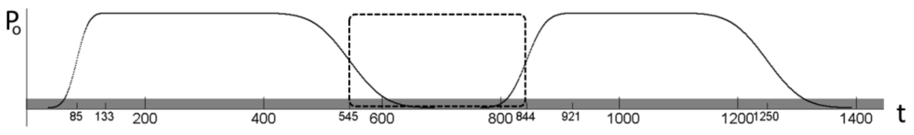

Fig. 2. Example of a resource occupation probability schedule.

For example, consider a resource availability probability for an interval T: [545; 844] presented as a dotted rectangle in Fig 2. Three independent events are active during the interval: 1) resource release event e_1 with the expected release time at 545 time units, 2) resource allocation event e_2 with the expected allocation time at 844 time units, and 3) a global utilization event e_3 with a constant occupation probability $P_o = 0.06$ (related details were provided with a Fig.1 example). Corresponding partial occupation and availability probabilities are: $P_o^{max}(e_1) = 0.5$, $P_o^{max}(e_2) = 0.5$, $P_o^{max}(e_3) = 0.06$, while $P_a(e_1) = 0.5$, $P_a(e_2) = 0.5$, $P_a(e_3) = 0.94$. So, the total probability of the resource availability during the whole interval T is $P_a^r = 0.235$.

2.2 Parallel Job Scheduling and Group Dependencies

To execute a parallel job, a set of simultaneously available nodes (a *window*) should be allocated ensuring user requirements from the resource request. The resource request usually specifies number n of nodes required simultaneously for a time period T and a maximum available resources allocation budget C. The total cost of a window allocation is calculated as $C_W = \sum_{i=1}^{n} T * c_i$, where c_i is resource i price for a single time unit.

These parameters constitute a formal generalization for resource requests common among distributed computing systems and simulators [13, 14-16]. Period T of the resources acquisition is usually the same for all resources selected for a parallel job. Common allocation and release times ensure the possibility of inter-node communications during the whole job execution. In this way, the *total window availability* is a function of availability probabilities of all the selected resources during the considered time interval T. More formally, when a set of n resources is selected for a job, the total window availability P_a^{w} during the expected job execution interval can be estimated as a product of availability probabilities $P_a^{\mathrm{r_i}}$ of each *independent* window nodes:

$$P_a^w = \prod_i^n P_a^{r_i}. \tag{2}$$

Here $P_a^{\mathrm{r_i}}$ can be calculated for each resource by the algorithm described in Section 2.1. If any of the window nodes will be occupied during the expected job execution interval (i.e., $P_a^{r_i} = 0$), the whole parallel job will be postponed or even aborted. Therefore, in general, the window allocation procedure should consider *maximization of the total probability of availability* $P_a^{\mathrm{w}} \rightarrow$ max. Based on the model above the general statement of the window allocation problem is as follows: during a scheduling interval L allocate a subset of n nodes with performance $p_i \geq p$ for a time T, with common allocation and release times and a restriction C on the total allocation cost. As a target optimization criterion, we assume maximization of the whole window availability probability (2).

As we additionally showed in [14, 15], this *general problem can be reduced to the following task*: at a given time t, which defines the set and state of m available resources, allocate a subset of n nodes with a restriction C on their total allocation cost while performing maximization of their total availability probability (2). In [14, 15] we proposed several approaches to solve the problems above. However, their statement and solution assume *independence* of individual resources as well as their utilization events. That is why in (2) we calculate the total window availability as a product of the availability probabilities of its elements.

In a more general and realistic model, the resources and their utilization events *are not independent*. On the contrary, there are group dependencies between the resources' parameters. The most typical example of such a dependency is a result of a parallel job execution. When a parallel job is scheduled, a set of selected resources is allocated for a common period T. That is, all the selected resources will share allocation, occupation, and release times. So, they should be modeled with a common chain of *allocation-occupation-release* events. In another words, these resources have a *group dependency*.

Fig. 3 shows example of utilization events modeled for a parallel job, which requested three nodes. Red areas present resources' utilization probability for allocation and release events. As the exact allocation and release times are unknown, the corresponding occupation probabilities $P_o(t) < 1$. Green areas show execution event with the occupation probability $P_o = 1$. The main issue is that criterion (2) becomes inaccurate when applied to a resources' set with many internal group dependencies. For example, in Fig. 3 if we consider total availability probability of resources 1, 4 and 5 at time $t = 400$, criterion (2) will calculate it as a product $P_a^{\mathrm{w}} = P_a^1 * P_a^3 * P_a^4$. However, as these resources are used by the same parallel job (and have a common group dependency), their actual total availability probability $P_a^{\mathrm{w}} = P_a^1 = P_a^3 = P_a^4 \geq P_a^1 * P_a^3 * P_a^4$.

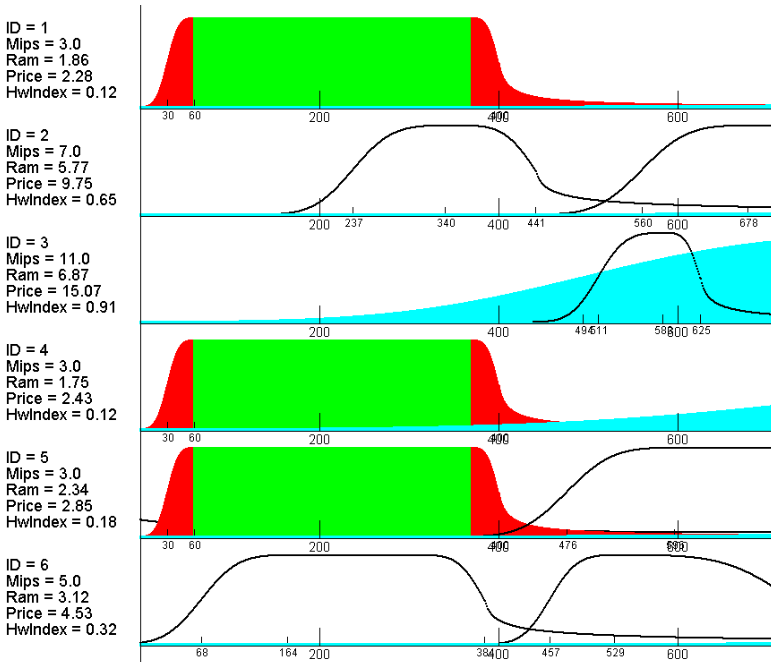

Fig. 3. Example of a parallel job execution schedule.

To describe it more formally we consider a set of groups G over the set R of the available resources. Each component group $G_i \in G$ represents a subset of resources $r_j \in R$ with a common group dependency. For example, one scheduled job, like in the example above, forms a single group G_i which includes all the resources selected for the job. So, for example, if one resource $r_j \in G_i$ is selected for a window W, the common group availability $P_a^{\mathrm{G_i}}$ should be used for calculation of a total W availability probability P_a^{w}. However, additionally selecting any other resources from G_i will not affect P_a^{w}, as their group probability component $P_a^{\mathrm{G_i}}$ is already considered.

So, the total window W availability probability can be calculated as follows:

$$P_a^{w} = \prod_{i}^{n^*} P_a^{G_i}, \tag{3}$$

where n^* is a number of diverse groups used for the window W, and $P_a^{G_i}$ is availability probability for each different group G_i used for the window. Group G_i is added to (3) if at least one of its resources is selected for the window. It is worth noting, that in the extreme case each group G_i can contain only one resource, and thus (3) will converge to (2). In this paper we propose and study resources allocation algorithm which performs (3) $P_a^w \to \max$ optimization considering economic constraint on the total window cost and group dependencies G. However firstly we should introduce helper algorithms performing (2) $P_a^w \to \max$ optimization without the group dependencies configuration.

2.3 Direct Solutions of the Resources Allocation Problem

Let us discuss in more details an algorithm which allocates an optimal (according to the probability criterion P_a^w) subset of n resources from the set R of m available resources with a limit C on their total cost.

Firstly, we consider maximizing the following total resources availability criterion $P_a^w = \prod_j^n p_a^{r_j}$, where $p_a^{r_j} = p_j$ is an availability probability of a single resource $r_j \in R$ during a considered interval T. In this way we can state the following problem of an n - size window subset allocation out of m nodes:

$$P_a^w = \prod_j^m x_j p_a^{r_j} \to \max,\ \sum_j^m x_j\, c_j \le C,\, x_j \in \{0,1\}, j = 1..m, \sum_j^m x_j = n, \tag{4}$$

where c_j is total cost required to allocate resource r_j, x_j - is a decision variable determining whether to allocate resource r_j ($x_j = 1$) or not ($x_j = 0$) for the current window.

This problem relates to the class of integer linear programming problems, and we used 0-1 knapsack problem as a base for our implementation. The classical 0-1 knapsack problem with a total weight C and items-resources with weights c_j and values p_j have a similar formal model except for extra restriction on the number of items required: $x_1 + x_2 + \cdots + x_m = n$. Therefore, we implemented the following dynamic programming recurrent scheme:

$$f_j(c, v) = \max\{f_{j-1}(c, v), f_{j-1}(c - c_j, v - 1)\ *\ p_j\}, \tag{5}$$

$$j = 1, .., m, c = 1, .., C, v = 1, .., n,$$

where $f_j(c, v)$ defines the maximum availability probability value for a v-size window allocated from the first j resources of m for a budget c. After the forward induction procedure (4) is finished the maximum availability value $P_a^w{}_{max} = f_m(C, n)$. x_j values are then obtained by a backward induction procedure.

An estimated computational complexity of the presented knapsack-based algorithm *KnapsackP* is $O(m * n * C)$.

Another approach for n-size window allocation is to use a more computationally efficient greedy approach. We outline four main greedy algorithms to solve the problem (3). The task is to select n out of m resources providing maximum total availability probability P_a^w with a constraint on their total allocation cost n.

1. *MaxP* selects first n nodes providing maximum availability probability p_j values. This algorithm does not consider total usage cost limit and may provide infeasible solutions. Nevertheless, *MaxP* can be used to determine the

best possible availability options and estimate a budget required to obtain them.

2. An opposite approach *MinC* selects first n nodes providing minimum usage cost c_j or an empty list in case of exceeding a total cost limit C. In this way, *MinC* does not perform any availability optimization, but always provides feasible solutions when it is possible. Besides, *MinC* outlines a lower bound on a budget required to obtain a feasible solution.
3. Third option is to use a weight function to regularize nodes in an appropriate manner. *MaxP/C* uses $w_j = p_j/c_j$ as a weight function and selects first n nodes providing maximum w_j values. Such an approach does not guarantee feasible solution, but nonetheless performs some availability optimization by implementing a compromise solution between *MaxP* and *MaxC*.
4. Finally, we consider a composite approach *GreedyUnited* for an efficient greedy-based resources allocation. The algorithm consists of three stages.
   a. Obtain *MaxP* solution and return it if the constraint on a total usage cost is met.
   b. Else, obtain *MaxP/C* solution and return it if the constraint on a total usage cost is met.
   c. Else, obtain *MinC* solution and return it if the constraint on a total usage cost is met.

   This combined algorithm *GreedyUnited* is designed to perform the best possible greedy optimization considering a restriction on a total resources' allocation cost C.

Estimated computational complexity for the greedy resources' allocation step is $O(m * \log m)$. More details regarding the algorithms above are provided in [14-16].

2.4 Resources Allocation Algorithms with Group Dependencies

Based on *KnapsackP* and *GreedyUnited* implementations above we propose the following algorithm for a general resource allocation problem considering group dependencies between the available resources. It takes as input set R of the available resources (each resource is characterized with cost c_i) and set G of groups over R (each group G_i has a common availability probability p_i). The algorithm then allocates a subset of n resources with a restriction C on their total cost while performing maximization of their total availability probability (3). The problem is solved by branch and bounds method by maintaining max-heap data structure H containing interim candidate solutions S_j. The higher the achieved availability probability P_a^w (3) or its upper bound, the closer the solution S to the top of the heap H. For each solution S we maintain two subsets of groups that should (G^+) and should not (G^-) be used in the current solution. Both G^+ and G^- are initialized as empty sets. Additionally, we consider subset G^0 as all groups from G not included in G^+ or G^-, so G^0 is initialized as G.

Initial candidate solution S^0 with empty $G^0 = G$ and empty sets G^+ and G^-, is placed into H with $P_a^w = -infinity$. Next, we perform the following steps.

1. Retrieve next solution candidate S from H. If S is marked as valid solution, then return S as a result, end of the algorithm.
2. Prepare list of resources R_s to calculate P_a^w for S.
   a. Init R_s as empty set.
   b. For each group G_j from G^+ add the cheapest resource to the solution window W_s with the $p_i = P_a^{G_j}$; add other resources from this group $r_i \in G_j$ to R_s with $p_i = 1$.
   c. For each group G_j from G^0 add all resources $r_i \in G_j$ to R_s with $p_i = \sqrt[k]{P_a^{G_j}}$, where k is number of resources in G_j.
3. Use algorithm *KnapsackP* or *GreedyUnited* to perform direct solution of S to allocate resources into W_s (it can be partially filled during step 2.b) from set R_s of prepared resources with (2) $P_a^w \rightarrow$ max optimization without group dependencies.
4. Check if the resulting solution is valid.
   a. If all resources from W_s are included in groups from G^+, then put this solution S into H with key P_a^w and mark it as a *valid solution*.
   b. If at least one resource r_s from W_s is included in some group G_s from G^0, then we need to split this solution S into two candidates: S^+ and S^-. For S^+ remove group G_s from G^0 and add into G^+. For S^- remove group G_s from G^0 and add into G^-. Put both solution candidates S^+ and S^- into H with key P_a^w as an upper estimate.
5. Go to step 1.

The algorithm above performs branch and bounds approach by splitting candidate solutions by sets of resources groups G^+ and G^- required to use or skip correspondingly. A special resource set R_s preparation in step 2 allows us to use (2) optimization algorithms and obtain either a final valid solution or a candidate solution with pretty accurate upper estimate. The algorithm finishes when the next solution obtained from the max-heap data structure is a valid solution composed of resources from G^+ groups and, thus, its P_a^w calculated with (2) satisfies rules for group dependencies availability calculations (3).

3 Simulation Study

3.1 Considered Algorithm Implementation

For the simulation study we consider and compare the following algorithm implementations.

1. Firstly, we implemented *brute-force* algorithm to solve the resources allocation problem with (3) $P_a^w \rightarrow$ max optimization. We used this algorithm for a preliminary analysis in small experiments with up to 21 resources to compare its optimization efficiency with other approaches.

2. Next, we prepared three implementations of a general branch and bounds algorithm described in Section 2.4. First implementation *KnapsackGroup* uses *KnapsackP* for all interim allocations during the algorithm step 3. *Greedy* performs interim optimizations at step 3 with *GreedyUnited* algorithm. Finally, *Greedy+* runs *GreedyUnited* for all interim optimizations, but once the solution is found, the final solution optimization is performed again using more accurate *KnapsackP* approach.
3. Finally, we consider *KnapsackP (KnapsackSingle)* as standalone algorithms for the comparison. This algorithm does not support group dependencies and performs (2) $P_a^w \rightarrow$ max optimization. The obtained solution is then recalculated accordingly to (3) to compare it to the algorithms above.

For the simulation study we execute and collect resulting data for all the considered algorithms (*BruteForce*, *KnapsackGroup*, *Greedy*, *Greedy+* and *KnapsackSingle)* in different resource environments with randomized characteristics c_i, p_i and group dependencies. An experiment was prepared using a custom distributed environment simulator [6, 14, 15]. For our purpose, it implements a heterogeneous resource domain model: nodes have different usage costs and availability probabilities. Each node supports a list of active global and local job utilization events. Fig. 3 shows an example of such an environment with many resources and a Gantt chart of the utilization events.

Additionally, we generate random uniformly distributed group dependencies between the resources. So, the resources allocation problem can be defined with the following parameters: N – number of available resources (each characterized with cost c_i and availability probability p_i), G – number of different groups (containing random non-intersecting subsets of resources), n – number of resources required for allocation and C – available budget, i.e. constraint on the total cost of the selected resources.

3.2 Proof of Optimization Efficiency

The first experiment series studies algorithms optimization and computational efficiency in comparison with *BruteForce* approach. Brute force is usually inapplicable in real-world tasks due to its exponential computational complexity. However, it guarantees exact optimization solution, and can be used to evaluate optimization characteristics of other considered algorithms. During each simulation experiment, the resources allocation was independently performed by algorithms *BruteForce*, *KnapsackGroup*, *Greedy*, *Greedy+* and *KnapsackSingle*. The comparison is obtained with different values of G, n, C of the allocation problem. As *BruteForce* applicability is limited, firstly we performed resources allocation simulation with only $N = 21$ available resources.

Fig.4 shows resulting availability probability P_a^w depending on number $n \in [1; 21]$ of requested resources in environment with $N = 21$ available resources, $G = 8$ different groups and without the total cost restriction ($C = \sum_i^N c_i$). The main result is that proposed algorithms *KnapsackGroup*, *Greedy* and *Greedy+* provided the same P_a^w value as *BruteForce* (that is why they are not presented in Fig. 4). *KnapsackGroup* theoretically guarantees exact problem solution in integers and is expected to provide results identical to *BruteForce*. Greedy algorithms provided optimal solution due to the lack of the total cost limit (see *GreedyUnited* and *MaxP* descriptions in Section 2.3). However, *KnapsackSingle* in most cases failed to provide optimal solution with up to 5% lower availability probability compared to *BruteForce*. The equality is achieved only in two simplified scenarios with $n = 1$ and $n = 21$, when group dependencies are not relevant for the problem.

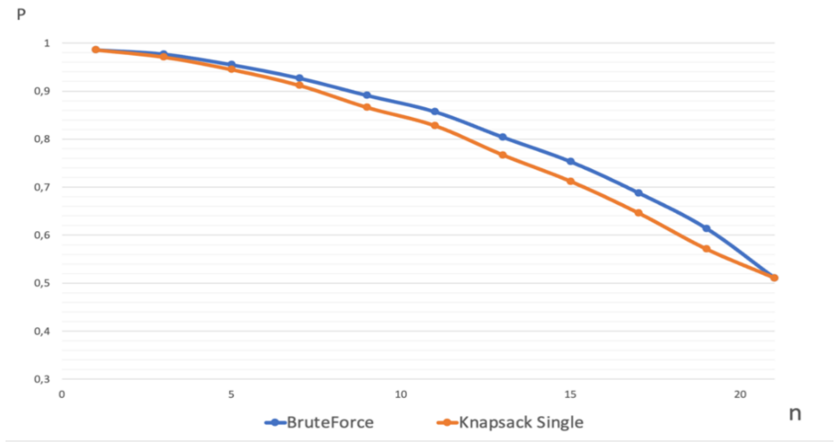

Fig. 4. Simulation results: resulting availability probability P_a^w depending on number n of requested resources.

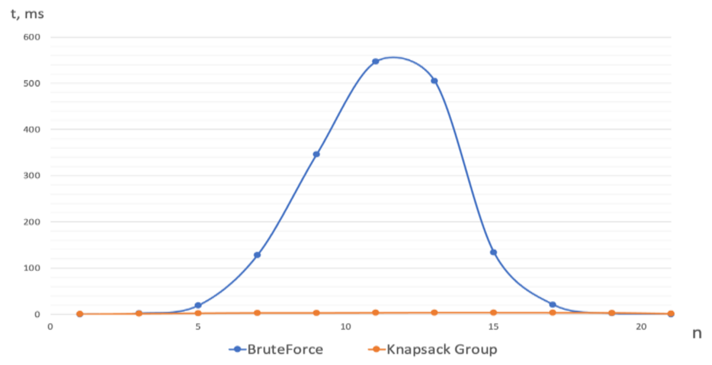

Fig. 5. Simulation results: average calculation time depending on number n of requested resources.

Fig. 5 shows actual algorithms' execution time required to achieve allocation results from Fig. 4. As can be seen, *BruteForce* calculation time dramatically increases for $n \in [7; 15]$ and exceeds half a second for $n = 11$. This is explained by the combinatorial nature of selecting subset of n from N available resources. Even the most computationally complex *KnapsackGroup* algorithm, which combines pseudo polynomial 0-1 knapsack implementation with branch and bounds approach is presented in Fig.5 as a straight line 100 times lower compared to the *BruteForce* maximum. *Greedy* approaches were up to 1000 times faster than *BruteForce*. So, according to the trend in Fig. 5, in environments with $N > 25$ *BruteForce* becomes practically inapplicable and other exact algorithms and approximations should be considered. The accuracy of such approximations in general should be estimated with the economical restriction C on the total window allocation cost.

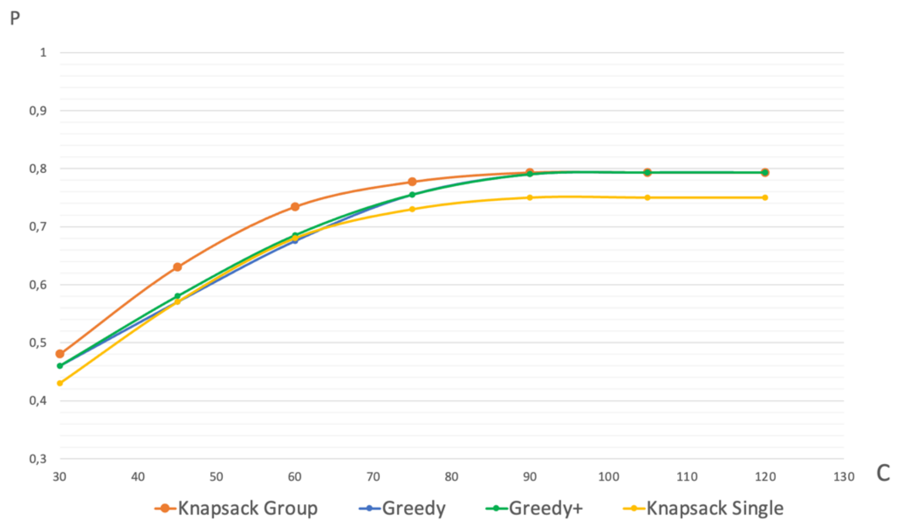

Fig. 6. Simulation results: resulting availability probability P_a^w depending on the budget C.

Fig.6 shows how window availability probability depends on the allocation budget $C \in [30; 120]$ in problem setup with $n = 8$, $G = 8$ and $N = 21$. In this environment only *KnapsackGroup* was able to obtain exact solutions (identical to *BruteForce*) for all C values. Additionally, *KnapsackGroup* provides almost constant 5% advantage over *KnapsackSingle*. The results of Greedy algorithms are also within 5% of the exact solution and reaches *BruteForce* for $C > 90$. In general, the obtained simulation result confirms accuracy of *KnapsackGroup* algorithm and gives an approximate estimate of the accuracy of the more computationally simple algorithms.

3.3 Practical Optimization Efficiency Study

Next experiment series studies proposed algorithms in more complex problem settings with $N = 200$, $G = 40$ and $n = 20$. As brute force becomes impractical for such figures, we use *KnapsackGroup* as a reference and accurate solution of (3).

Firstly, Fig. 7 shows availability probability as a function of $C \in [40; 220]$. Lower bound was selected so that it was almost impossible just to allocate any 20 resources with budget $C < 40$, without any optimization. So, the resulting P_a^w generally increase with increasing C. Upper bound $C > 200$ allows to select almost any resources without checking for the total cost limit. In this experiment setup with more resources and optimization variability, Greedy algorithms are already seriously losing the accuracy of the solution. The advantage of *KnapsackGroup* exceeds 20% for some values of C. This result generally correlates with works [14, 15]. At the same time, *KnapsackSingle* provides availability probability only 10% lower than the exact solution. In this way, the absence of group dependencies information turns out advantageous compared to the accuracy of greedy approximations of the multiplicative knapsack problem. Only in scenarios with $C > 200$, i.e., without the cost restriction, *Greedy* can outperform *Knasack Single* in environment with group dependencies between the resources.

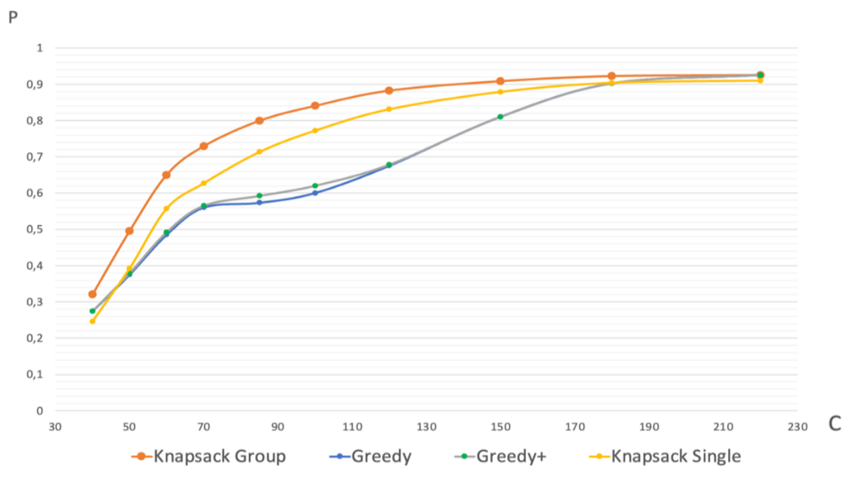

Fig. 7. Simulation results: resulting availability probability P_a^w depending on the budget C.

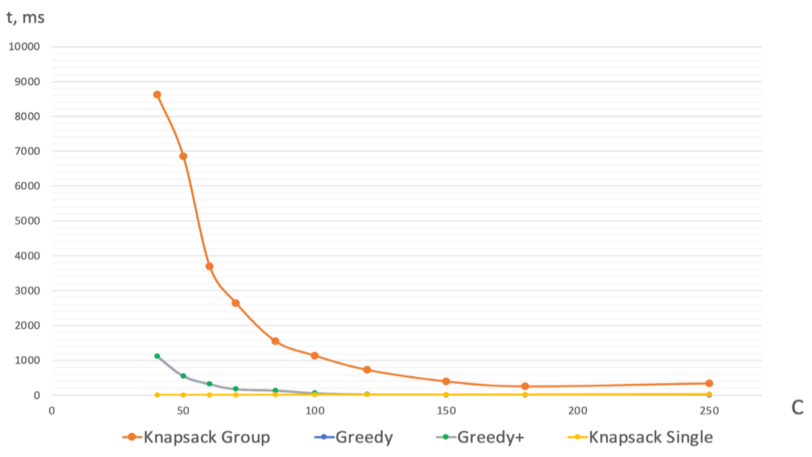

Fig. 8. Simulation results: average calculation time depending on the budget C.

Another crucial factor for the practical applicability is the algorithms' calculation time presented in Fig. 8 for the same environment settings. *Greedy* and *Greedy+* have almost identical calculation times and so their graphs fully overlap. The obvious trend is that tighter restrictions on the budget C cause a strong increase in working time for branch and bounds - based algorithms (*KnapsackGroup*, *Greedy*, *Greedy*+). This is explained by the necessity to select resources with respect to the C constraint, rather than by the target criterion. And this strategy requires consideration of more diverse groups and splitting in branch and bounds approach. For example, with $C = 40$, an average size of the solution tree for *KnapsackGroup* was almost 5000 elements causing nearly 8 seconds of the execution time. And with $C = 120$ the tree size decreased to nearly 100 elements leading to a sub second execution time. Similar calculation time trend applies to Greedy tree algorithms as well.

Thus, based on Figs. 7, 8 we conclude, that with tight economical budget restrictions the most practically adequate option is a simple multiplicative 0-1 knapsack algorithm [14], as such problem setup requires greater emphasis on the cost optimization and less on the groups' combinations. With a looser cost restriction ($C \in [100; 200]$ in our experiment) tree-based *KnapsackGroup* becomes a preferred option as it provides exact optimization solution for an adequate calculation time. Finally, when there is no cost restriction, a tree-based *Greedy* algorithm can provide exact optimization result in the least amount of calculation time.

4 Conclusion and Future Work

In this work, we address the problem of dependable resources co-allocation for parallel jobs in distributed computing with group dependencies over the resources. Such group dependencies usually define utilization events common for subsets of resources, such as simultaneous allocation or release events. To handle this problem, we designed several branches and bounds algorithms based on a multiplicative 0-1 knapsack problem.

In a simulation study we proved accuracy of the proposed algorithms in comparison with a brute force approach, estimated their calculation time and practical applicability in a more complex scheduling problems with up to 200 available computing nodes.

Future work will concern additional optimization in the algorithms' complexity and calculation time. In addition, we plan to consider similar allocation task based on an additive 0-1 knapsack problem.

Acknowledgments. This work was supported by the Russian Science Foundation (project No. 22-21-00372, https://rscf.ru/en/project/22-21-00372/).

References

1. Lee, Y.C., Wang C., Zomaya, A.Y., Zhou, B.B.: Profit-driven Scheduling for Cloud Sevices 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).

3. 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).
4. 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).
5. Kurowski, K., Nabrzyski, J., Oleksiak, A., Weglarz, J.: 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, pp. 271-293 (2003).
6. Toporkov, V., Toporkova, A., Bobchenkov, A., Yemelyanov, D.: Resource Selection Algorithms for Economic Scheduling in Distributed Systems. ICCS 2011, June 1-3, 2011, Singapore, Procedia Computer Science. Elsevier, vol. 4. pp. 2267-2276 (2011).
7. Netto, M. A. S., Buyya, R.: A Flexible Resource Co-Allocation Model based on Advance Reservations with Rescheduling Support. In: Technical Report, GRIDSTR-2007-17, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, October 9, 2007.
8. Jackson, D., Snell, Q., Clement, M.: Core Algorithms of the Maui Scheduler. 7th International Workshop on Job Scheduling Strategies for Parallel Processing, pp. 87-102 (2001).
9. Javadi, B., Kondo, D., Vincent, J., Anderson, D.: Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home. IEEE Transactions on Parallel and Distributed Systems 22 (11), 1896 - 1903 (2011).
10. Rood, B., Lewis, M.J. Grid Resource Availability Prediction-Based Scheduling and Task Replication. J. Grid Computing 7, 479 (2009).
11. Tchernykh, A., Schwiegelsohn, U., El-ghazali, T., Babenko, M.: Towards Understanding Uncertainty in Cloud Computing with Risks of Confidentiality, Integrity, and Availability, J. Comput. Sci. vol.36. (2016).
12. Chaari, T., Chaabane, S., Aissani, N., and Trentesaux, D.: Scheduling Under Uncertainty: Survey and Research Directions. 2014 Int. Conf. on Advanced Logistics and Transport (ICALT), pp. 229-234 (2014).
13. 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).
14. Toporkov, V., Yemelyanov, D.: Availability-based Resources Allocation Algorithms in Distributed Computing. In: Voevodin, V., Sobolev, S. (eds) Supercomputing. CCIS, vol. 1331, pp. 551–562. Springer, Cham (2020).
15. Toporkov, V., Yemelyanov, D., Grigorenko, M.: Optimization of Resources Allocation in High Performance Computing Under Utilization Uncertainty. In: Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds) Computational Science – ICCS 2021. Lecture Notes in Computer Science, vol. 12747, pp. 540–553 Springer, Cham (2021).
16. Toporkov, V., Yemelyanov, D., Bulkhak, A. Machine Learning-Based Scheduling and Resources Allocation in Distributed Computing. In: Groen, D., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds) Computational Science – ICCS 2022. ICCS 2022. Lecture Notes in Computer Science, vol. 13353, pp. 3–16. Springer, Cham (2022).