Anticipation Scheduling in Grid with Stakeholders Preferences

Victor Toporkov[1], Dmitry Yemelyanov[1(✉)], and Anna Toporkova[2]

[1] National Research University "MPEI", Moscow, Russia
{ToporkovVV,YemelyanovDM}@mpei.ru
[2] National Research University Higher School of Economics, Moscow, Russia
atoporkova@hse.ru

Abstract. In this work, a job-flow scheduling approach for grid virtual organizations (VOs) is proposed and studied. Users' and resource providers' preferences, VOs internal policies, resources geographical distribution along with local private utilization impose specific requirements for efficient scheduling according to different, usually contradictive, criteria. With increasing level of resources utilization, the set of available resources and corresponding decision space are reduced. This further complicates the problem of efficient scheduling. In order to improve overall scheduling efficiency, we propose an anticipation scheduling approach based on a cyclic scheduling scheme. It generates a near optimal but infeasible scheduling solution and includes a special replication procedure for efficient and feasible resources allocation. Anticipation scheduling is compared with the general cycle scheduling scheme and conservative backfilling using such criteria as average jobs' start and finish times as well as users' and VO economic criteria: total execution time and cost.

Keywords: Scheduling · Grid · Resources · Utilization · Heuristic · Job batch · Virtual organization · Cycle scheduling scheme · Anticipation · Replication

1 Introduction and Related Works

In grids with non-dedicated resources the computational nodes are usually partly utilized by local high-priority jobs coming from resource owners. Thus, the resources available for use are represented with a set time intervals (slots) during which the individual computational nodes are capable to execute parts of independent users' parallel jobs. These slots generally have different start and finish times and a performance difference. The presence of a set of slots impedes the problem of resources allocation necessary to execute the job flow from VOs users. Resource fragmentation also results in a decrease of the total level of computing environment utilization [1, 2].

Application-level scheduling [3], as a rule, does not imply any global resource sharing or allocation policy. Applications try to control grid resources independently. Job flow scheduling in VOs [4, 5] supposes uniform rules of resource sharing and consumption, in particular based on economic models [2, 4–6]. Usually there are three parties in these models: users, resource owners, and VO administrators. General

V. Voevodin and S. Sobolev (Eds.): RuSCDays 2017, CCIS 793, pp. 482–493, 2017.
https://doi.org/10.1007/978-3-319-71255-0_39

interaction and resources or services provisioning between these parties is performed by means of a certain currency. VO scheduling policy may offer optimization rules to satisfy both users' and VO common preferences (owners' and administrators' combined). The VO scheduling problems may be formulated as follows: to optimize users' criteria or utility function for selected jobs [6, 7], to keep resource overall load balance [8, 9], to have job run in strict order or maintain job priorities [10], to optimize overall scheduling performance by some custom criteria [11, 12], etc.

Users' preferences and VO common preferences may conflict with each other. Users are likely to be interested in the fastest possible running time for their jobs with least possible costs whereas VO preferences are usually directed to balancing of available resources load or node owners' profit boosting. In fact, an economical model of resource distribution per se reduces tendencies to cooperate [13]. Thus, VO economic policies in general should respect all members to function properly and the most important aspect of rules suggested by VO is their fairness. A number of works understand fairness as it is defined in the theory of cooperative games [7], such as fair job flow distribution [9], fair quotas [14, 15], fair user jobs prioritization [10], and non-monetary distribution [16]. In many studies VO stakeholders' preferences are usually ensured only partially: either owners are competing for jobs optimizing only users' criteria [6, 17], or the main purpose is the efficient resources utilization not considering users' preferences [18].

The goal of the current study is to design a general job-flow scheduling approach which will be able to find a tradeoff between VO stakeholders' contradictory preferences based on the cyclic scheduling scheme (CSS). CSS [19, 20] has fair resource share in a sense that every VO stakeholder has mechanisms to influence scheduling results providing own preferences. Thus, we elaborate a problem of parallel jobs scheduling in heterogeneous computing environment with non-dedicated resources considering users' individual preferences and goals.

The downside of a majority of centralized metascheduling approaches is that they lose their efficiency and optimization features in distributed environments with a significant workload. In such conditions of a *limited resources supply* overall job-flow execution makespan and individual jobs' finish time minimization become essential scheduling criteria. For example in [2], a traditional backfilling algorithm provided better scheduling outcome when compared to different optimization approaches in resource domain with a minimal performance configuration.

Main contribution of this paper is a CSS-based heuristic *anticipation* approach which retains scheduling efficiency and at the same time minimizes job-flow processing time. Initially this heuristic generates a near optimal but infeasible (anticipated) schedule. A special *replication* procedure is proposed and studied to ensure and provide a feasible scheduling solution.

The rest of the paper is organized as follows. Section 2 presents a general CSS fair scheduling concept. The proposed heuristic-based scheduling technique is presented in Sect. 3. Section 4 contains experiment setup and results for the proposed scheduling approach and its comparison with backfilling. Finally, Sect. 5 summarizes the paper.

2 Cyclic Alternative-Based Scheduling

Scheduling of a job flow using CSS is performed in time cycles known as scheduling intervals, by job batches [19, 20]. The actual scheduling procedure consists of two main steps. The first step involves a search for alternative scenarios of each job execution, or simply alternatives [21]. During the second step the dynamic programming methods [19, 20] are used to choose an optimal alternatives' combination. One alternative is selected for each job with respect to the given VO and user criteria. An example for a user scheduling criterion may be a job runtime, finish time, an overall running cost, etc. This criterion describes user's preferences for that specific job execution and expresses a type of an additional optimization to perform when searching for alternatives. Alongside with time (T) and cost (C) properties each job execution alternative has a user utility (U) value: user evaluation against the scheduling criterion. A common VO optimization problem may be stated as either minimization or maximization of one of the properties, having other fixed or limited, or involve Pareto-optimal strategy search involving both kinds of properties [3, 20, 22].

We consider the following relative approach to represent the user utility U. A job alternative with the minimum (best) user-defined criterion value $Z_{\min}$ corresponds to the left interval boundary ($U = 0\%$) of all possible job scheduling outcomes. An alternative with the worst possible criterion value $Z_{\max}$ corresponds to the right interval boundary ($U = 100\%$). In the general case, for each alternative with value Z, U is set depending on its position in $[Z_{\min}; Z_{\max}]$ interval as follows: $U = \frac{Z - Z_{\min}}{Z_{\max} - Z_{\min}} * 100\%$. Thus, each alternative gets its utility in relation to the "best" and the "worst" optimization criterion values user could expect according to the job's priority. The more some alternative corresponds to user's preferences the smaller is the U value.

For a fair scheduling model the second step of the VO optimization problem could be in form of: $C \rightarrow \max$, $\lim U$ (maximize total job flow execution cost, while respecting user's preferences to some extent: $U \leq U_{\max}$); $U \rightarrow \min$, $\lim T$ (meet user's best interests, while ensuring some acceptable job flow execution time: $T \leq T_{\max}$) and so on [19].

The launch of any job requires a co-allocation of a specified number of slots, as well as in the classic backfilling variation. A single slot is a time span that can be assigned to run a part of a parallel job. The target is to scan a list of available slots and to select a window of parallel slots with a "length" of the required resource reservation time. The user job requirements are arranged into a resource request containing a resource reservation time, characteristics of computational nodes (clock speed, RAM volume, disk space, operating system etc.), limitation on the selected window maximum cost.

ALP, AMP and AEP window search algorithms were discussed in [21]. The job batch scheduling performs consecutive allocation of a multiple nonintersecting in terms of slots alternatives for each job. Otherwise irresolvable collisions for resources may occur if different jobs will share the same time-slots. Sequential alternatives search and resources reservation procedures help to prevent such scenario. However in an extreme case when resources are limited or over utilized only at most one alternative execution could be reserved for each job. In this case alternatives-based scheduling result will be

no different from First Fit resources allocation procedure [2]. First Fit resource selection algorithms [23] assign any job to the first set of slots matching the resource request conditions without any optimization.

3 Cyclic Anticipation Scheduling

In order to address the scheduling optimization problem the following anticipation heuristic for job batch scheduling is proposed. It consists of three main steps.

First, a set of all possible execution alternatives is found for each job not considering time slots intersections and without any resources reservation. The resulting intersecting alternatives found for each job reflect a full range of different job execution possibilities which user may expect on the current scheduling interval.

Second, CSS procedure [19, 20] is performed to select alternatives combination (one alternative for each job of the batch) optimal according to VO policy. The resulting alternatives combination most likely corresponds to an infeasible scheduling solution as possible time slots intersection will cause collisions on resources allocation stage. The main idea of this step is that obtained infeasible and *anticipated* solution will provide some *heuristic insights* on how each job should be handled during the scheduling. For example, if time-biased or cost-biased execution is preferred, how it should correspond to user criterion and VO administration policy and so on.

Third, a feasible resources allocation is performed. The resulting solution is both feasible and efficient as it reflects scheduling pattern obtained from a near-optimal reference solution – a *replication* step. The base for this replication is an Algorithm searching for Extreme Performance (AEP) described in details in [21]. AEP helps to find and reserve feasible execution alternatives most similar to those selected in the near-optimal infeasible solution.

We used AEP modification to allocate a diverse set of execution alternatives for each job. Originally AEP scans through a whole list of available time slots and retrieves one alternative execution satisfying user resource request and optimal according to the user custom criterion. During this scan, we saved all intermediate AEP search results to a dedicated list of possible alternatives.

For the replication purpose a new Execution Similarity criterion was introduced which helps AEP to find a window with a minimum distance to a reference alternative. Generally, we define a distance between two different alternatives (windows) as a relative difference or error between their significant criteria values. For example if reference alternative has C_{ref} total cost, and some candidate alternative cost is C_{can}, then the relative cost error E_C is calculated as $E_C = \frac{|C_{\text{ref}} - C_{\text{can}}|}{C_{\text{ref}}}$. If one needs to consider several criteria the distance D between two alternatives may be calculated as a linear sum of criteria errors: $D_l = E_C + E_T + .. + E_U$, or as a geometric distance in a parameters space: $D_g = \sqrt{E_C^2 + E_T^2 + .. E_U^2}$.

AEP modification with the Execution Similarity criterion is represented below.

```
Input Data:
    slotList - a list of available slots ordered non-
    decreasingly by their start time;
    job - a job for which the search is performed;
    refAlternative – reference alternative used to find similar
    job execution window.

Result:
    closestWindow – execution window similar to refAlternative

begin
  minDistance = MAX_VALUE;

  for each slot in slotList do
    if not(properHardwareAndSoftware(job, slot.node))
      continue;
    end if;

    windowSlotList.add(slot);
    windowStartTime = slot.startTime;

    for each wSlot in windowSlotList do
      minLength = wSlot.node.getWorkingTimeEstimate();
      if ((wSlot.endTime - windowStartTime) < minLength)
        windowSlotList.remove(wSlot);
      end if;
    end for;

    if (windowSlotList.size() ≥ job.nodesNeed)
      distance = calculateDistance(windowSlotList, refAlterna-
    tive);
      if (distance < minDistance)
        minDistance = distance;
        closestWindow = windowSlotList;
      end if;
    end if;
  end for;
end
```

In this algorithm an expanded window *windowSlotList* moves through a whole list of all available slots *slotList* sorted by their start time in ascending order. At each step any combination of *job.nodesNeed* slots inside *windowSlotList* can form a window that meets all the requirements to run the job. The main difference from the original AEP is that instead of searching for a window with a maximum single criterion value, we

retrieve window with a minimum distance D_g or D_l to a reference execution alternative. Generally, this distance can reflect job execution preferences in terms of multiple criteria such as job execution cost, runtime, start time, finish time, etc.

4 Simulation Study

An experiment was prepared as follows using a custom distributed environment simulator [2, 19–21]. 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 [24]) and, thus, each node can process only one task at any given simulation time. The execution cost of each task depends on its execution time which is proportional to the dedicated node's performance level. The execution of a single job requires parallel execution of all its tasks.

The simulation environment was configured with the following features. The resource pool includes 80 heterogeneous computational nodes grouped in a single resource domain. A specific cost of a node is an exponential function of its performance value (base cost) with an added variable margin distributed normally as ± 0.6 of a base cost. The scheduling interval length is 800 time quanta. The initial resource load with owner jobs is distributed hyper-geometrically resulting in 5% to 10% time quanta excluded in total.

Jobs number in a batch is 75. Nodes quantity needed for a job is a whole number distributed evenly on [2; 6]. Node reservation time is a whole number distributed evenly on [100; 500]. Job budget varies in the way that some of jobs can pay as much as 160% of base cost whereas some may require a discount. Every request contains a specification of a custom user criterion which is one of the following: job execution runtime or overall execution cost.

4.1 Replication Scheduling Accuracy

The first experiment is dedicated to a replication scheduling accuracy study. For this matter we conducted and collected data from more than 1000 independent job batch scheduling simulations. First, the general CSS was performed in each experiment for the following job-flow execution cost maximization problem $C \rightarrow \max$, $\lim U_a = 10\%$. U_a stands for the average user utility for one job, i.e. $\lim U_a = 10\%$ means that at average resulting deviation from the best possible outcome for each user did not exceed 10%. Next, linear and geometric replication algorithms were executed to replicate CSS solution using linear D_l and geometric D_g distance criteria. In the current experiment we used job execution cost error and processor time usage error to calculate distances.

In order to evaluate the resulting difference in scheduling outcomes, we additionally performed CSS algorithm ensuring users' individual preferences only ($\lim Ua = 0\%$) and ensuring VO preference by maximizing overall cost without taking into account users' criteria ($\lim Ua = 100\%$). These additional problems reflect extreme boundaries for scheduling results, which can be used to evaluate a relative replication error. Table 1 contains scheduling results for all these three problems and two replication algorithms.

Table 1. CSS replication average scheduling results

Job execution characteristic	C -> max, lim $Ua = 0\%$	C -> max, lim $Ua = 10\%$	Linear replication	Geometric replication	C -> max, lim $Ua = 100\%$
Cost	1283	***1349***	1353	1353	1475
Processor time	191.6	***191.2***	190.6	190.5	202.3
Finish time	367.1	***353.8***	356.2	356.4	358.5
U_a, %	0	***9.9***	17.6	17.8	65

The results indicate that both linear and geometric replication algorithms provided average scheduling parameters very close to the reference solution (indicated as bold in Table 1), and especially close against job execution cost and processor time usage, i.e. characteristics which were used for a replication distance calculation. For example, borderline problems provided average job execution cost (main job-flow optimization criterion) values 1283 and 1475 correspondingly. Reference intermediate solution provided 1349. And both replication algorithms ensured average job execution cost 1353 with only 2% deviation from reference solution against [1283; 1475] interval of possible scheduling outcomes. Although replication algorithms showed their efficiency with respect to integral job flow processing parameters (such as average job execution cost, runtime, finish time), individual user's preferences were considered to a lesser extent. It can be observed in the Table 1 that both replication algorithms provided average user utility U_a almost twice as much as the reference problem.

4.2 Anticipation and Backfilling Scheduling Comparison

The second experiment setup reiterates work [2] and is intended to compare anticipation scheduling procedure with a traditional backfilling algorithm. Backfilling is able to minimize the whole job-flow execution makespan as well as to generally follow the initial jobs relative queue order. These features make backfilling scheduling solution a good reference target for the anticipation scheduling scheme. The main criteria for comparison include average jobs' start and finish times as well as users' and VO economic criteria (such as execution time and cost). We used the following three algorithms for the comparison:

- CSS – the original cycle scheduling scheme;
- ANT – the anticipation scheduling procedure;
- BF – the conservative backfilling algorithm.

In a single experiment CSS and ANT solved $C \rightarrow \max$, lim $U_a = 10\%$ problem. Execution cost ($C \rightarrow \min$) and processor time ($T \rightarrow \min$) criteria were uniformly distributed between 75 user jobs generated in each experiment.

Important addition was introduced for ANT scheduling. In contrast with experiment series in Subsect. 4.1, job replication geometric distance D_g was calculated as $D_g = \sqrt{E_C^2 + E_T^2 + E_S^2}$, where additional element E_s stands for job start time error. As a reference start time value for each job we used start time obtained for a particular job by a prior backfilling scheduling. Thus, when searching for a job execution window we used infeasible solution for time and cost reference values, and a feasible backfilling solution as a reference for an attainable start time values complying with a queue priority.

To observe the behavior of the main scheduling parameters we conducted experiments with a different number N of computing nodes available during the scheduling: $N \in \{20, 25, 30, 40\}$.

Average job's start and finish times are presented in Figs. 1 and 2.

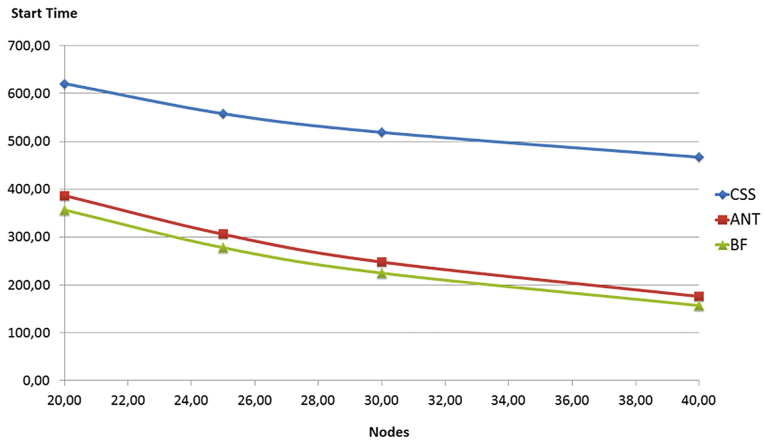

Fig. 1. Average jobs' start time in $C \rightarrow \max$, $\lim U$ problem

As can be seen in Figs. 1 and 2, backfilling provided better start and finish times for a job-flow execution compared to CSS and this result is consistent with [2]. In the current problem setup backfilling was able to finish the job flow execution almost twice earlier then CSS. It can be explained by $C \rightarrow \max$, $\lim U$ scheduling problem which required CSS to allocate resources for job-flow execution cost maximization considering contradictory user preferences, not minimizing jobs' completion times.

At the same time anticipation algorithm during each experiment solved the same $C \rightarrow \max$, $\lim U$ problem and provided jobs' start and finish times only 10% behind the backfilling scheduling outcome.

The details of anticipation scheduling can be examined in Figs. 3 and 4.

Figure 3 shows average job execution time provided by backfilling and anticipation algorithm. Additionally ANT T and ANT C represent average execution times obtained by anticipation scheduling for jobs with time minimization and cost minimization criteria correspondingly. As it can be observed, ANT and BF generally provided comparable execution times, which is not a direct optimization criterion for either of

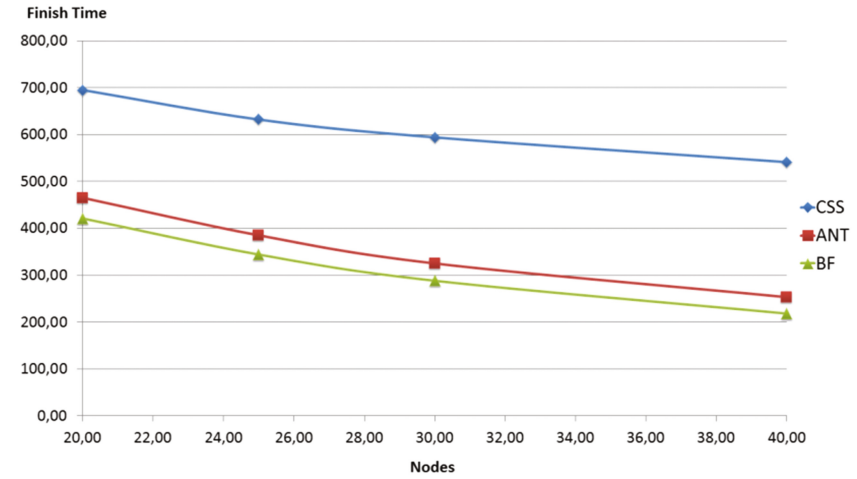

Fig. 2. Average jobs' finish time in $C \rightarrow \max$, $\lim U$ problem

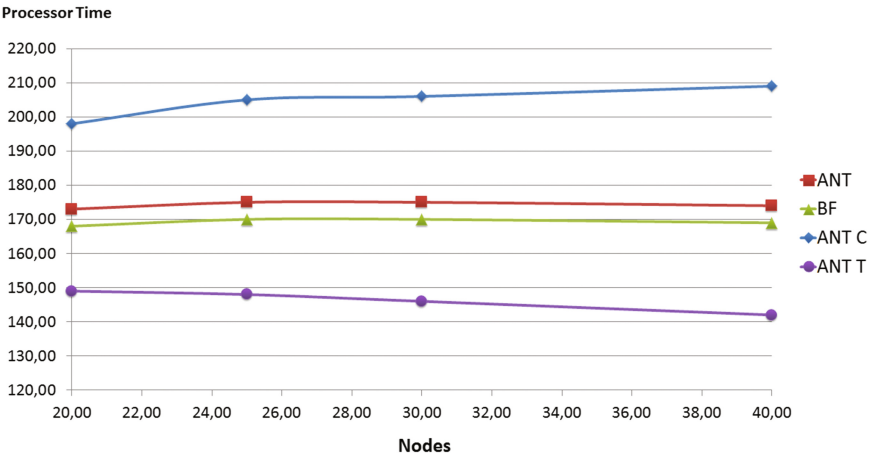

Fig. 3. Average jobs' execution time in $C \rightarrow \max$, $\lim U$ problem

them. At the same time ANT applied completely different scheduling policies for jobs with different private scheduling criteria. So that ANT T jobs used 25%–33% less processor time then ANT C jobs and 15% less compared to BF solution.

A similar pattern can be observed in Fig. 4, where average jobs' execution cost is presented. ANT and BF provided comparable general job-flow execution cost value. However ANT was able to consider user preferences and shared resources so that ANT C jobs execution cost was 10–15% less then ANT T jobs and 6–9% less compared to backfilling.

Summarizing the results, ANT is able to provide a general scheduling outcome similar to backfilling (with at most 10% error on job's start and finish times), and at the

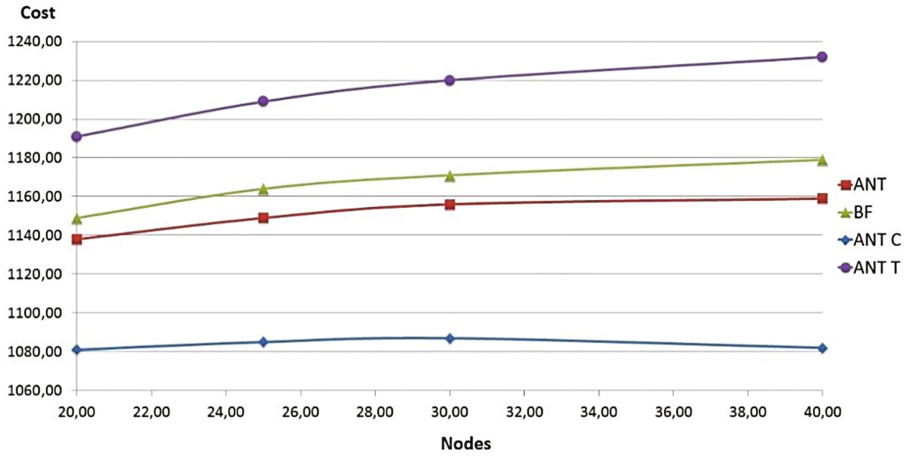

Fig. 4. Average jobs' execution cost in $C \rightarrow \max$, $\lim U$ problem

same time considers users' and VO preferences by efficiently solving $C \rightarrow \max$, $\lim U$ problem. Thereby the available resources are distributed between user jobs according to the predefined scheduling requirements (see Figs. 3 and 4). In our experiment set they include individual jobs execution preferences (for example, certain job's execution cost minimization) and a common job-flow scheduling policy (total job-flow execution cost maximization in our example).

Speaking of a whole job-flow scheduling policy it is worth noting that despite the cost maximization performed by ANT, backfilling still provided higher total job-flow execution cost (Fig. 4). This result may be explained by the need of ANT to additionally consider user preferences ($\lim U_a = 10\%$), including user jobs with a cost *minimization* criterion. For example, in $C \rightarrow \max$, $\lim U_a = 100\%$ problem, which performs cost maximization without taking into account user preferences, ANT provides 1–2% higher job-flow execution cost compared to backfilling, but does not reach original CSS by 10%. In this case ANT was limited by a start time reference (obtained from backfilling solution) and, thus, had fewer opportunities to use available resources for a total cost maximization as opposed to CSS.

5 Conclusions and Future Work

In this paper, we study the problem of fair job batch scheduling with a relatively limited resources supply. The main problem that arises is a scarce set of job execution alternatives which eliminates scheduling optimization efficiency.

We propose a heuristic anticipation scheduling which generates a near-optimal but infeasible reference solution and then replicates it to allocate a feasible accessible solution. The special replication procedure is proposed which provides 2–5% error from the reference scheduling solution. The obtained results show that the new heuristic approach provides flexible and efficient solutions for different fair scheduling

scenarios. In case when computing environment with a limited set of resources is considered the anticipation algorithm is still able to allocate resources according to VO stakeholders' preferences, generally complies with queue priorities and provides a job-flow completion time up to 10% behind backfilling solution.

Future work will be focused on replication algorithm studies and its possible application to fulfill complex user preferences expressed in a resource request. Reference parameters may be obtained from user expectations or transformed from different scheduling solutions. Different weights may be introduced for errors calculation on different reference parameters.

Acknowledgments. This work was partially supported by the Council on Grants of the President of the Russian Federation for State Support of Young Scientists and Leading Scientific Schools (grants YPhD-2297.2017.9 and SS-6577.2016.9), RFBR (grants 15-07-02259 and 15-07-03401), and by the Ministry on Education and Science of the Russian Federation (project no. 2.9606.2017/8.9).

References

1. Dimitriadou, S.K., Karatza, H.D.: Job scheduling in a distributed system using backfilling with inaccurate runtime computations. In: Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems, pp. 329–336 (2010)
2. Toporkov, V., Toporkova, A., Tselishchev, A., Yemelyanov, D., Potekhin, P.: Heuristic strategies for preference-based scheduling in virtual organizations of utility grids. J. Ambient Intell. Humanized Comput. **6**(6), 733–740 (2015)
3. 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, pp. 271–293. Kluwer Academic Publishers (2003)
4. Buyya, R., Abramson, D., Giddy, J.: Economic models for resource management and scheduling in grid computing. J. Concurrency Comput. **14**(5), 1507–1542 (2002)
5. Rodero, I., Villegas, D., Bobroff, N., Liu, Y., Fong, L., Sadjadi, S.M.: Enabling interoperability among grid meta-schedulers. J. Grid Comput. **11**(2), 311–336 (2013)
6. 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). https://doi.org/10.1007/3-540-36180-4_8
7. Rzadca, K., Trystram, D., Wierzbicki, A.: Fair game-theoretic resource management in dedicated grids. In: IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007), Rio De Janeiro, Brazil, pp. 343–350. IEEE Computer Society (2007)
8. Vasile, M., Pop, F., Tutueanu, R., Cristea, V., Kolodziej, J.: Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. J. Future Gener. Comput. Syst. **51**, 61–71 (2015)
9. Penmatsa, S., Chronopoulos, A.T.: Cost minimization in utility computing systems. Concurrency Comput. Pract. Exp. **16**(1), 287–307 (2014). Wiley
10. Mutz, A., Wolski, R., Brevik, J.: Eliciting honest value information in a batch-queue environment. In: 8th IEEE/ACM International Conference on Grid Computing, New York, USA, pp. 291–297 (2007)

11. Blanco, H., Guirado, F., Lérida, J.L., Albornoz, V.M.: MIP model scheduling for multi-clusters. In: Caragiannis, I., et al. (eds.) Euro-Par 2012. LNCS, vol. 7640, pp. 196–206. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36949-0_22
12. 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). https://doi.org/10.1007/978-3-642-16505-4_2
13. Vohs, K., Mead, N., Goode, M.: The psychological consequences of money. Science **314** (5802), 1154–1156 (2006)
14. Carroll, T., Grosu, D.: Divisible load scheduling: an approach using coalitional games. In: Proceedings of the Sixth International Symposium on Parallel and Distributed Computing, ISPDC 2007, p. 36 (2007)
15. Kim, K., Buyya, R.: Fair resource sharing in hierarchical virtual organizations for global grids. In: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, Austin, USA, pp. 50–57. IEEE Computer Society (2007)
16. Skowron, P., Rzadca, K.: Non-monetary fair scheduling cooperative game theory approach. In: Proceeding of SPAA 2013 Proceedings of the Twenty-Fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 288–297. ACM, New York (2013)
17. Dalheimer, M., Pfreundt, F.-J., Merz, P.: Agent-based grid scheduling with Calana. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 741–750. Springer, Heidelberg (2006). https://doi.org/10.1007/11752578_89
18. Jackson, D., Snell, Q., Clement, M.: Core algorithms of the Maui scheduler. In: Feitelson, D. G., Rudolph, L. (eds.) JSSPP 2001. LNCS, vol. 2221, pp. 87–102. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45540-X_6
19. Toporkov, V., Yemelyanov, D., Bobchenkov, A., Tselishchev, A.: Scheduling in grid based on VO stakeholders preferences and criteria. In: Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J. (eds.) Dependability Engineering and Complex Systems. AISC, vol. 470, pp. 505–515. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39639-2_44
20. Toporkov, V., Toporkova, A., Tselishchev, A., Yemelyanov, D., Potekhin, P.: Metascheduling and heuristic co-allocation strategies in distributed computing. Comput. Inf. **34**(1), 45–76 (2015)
21. Toporkov, V., Toporkova, A., Tselishchev, A., Yemelyanov, D.: Slot selection algorithms in distributed computing. J. Supercomput. **69**(1), 53–60 (2014)
22. Farahabady, M.H., Lee, Y.C., Zomaya, A.Y.: Pareto-optimal cloud bursting. IEEE Trans. Parallel Distrib. Syst. **25**, 2670–2682 (2014)
23. Cafaro, M., Mirto, M., Aloisio, G.: Preference-based matchmaking of grid resources with CP-Nets. J. Grid Comput. **11**(2), 211–237 (2013)
24. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. J. Softw. Pract. Exp. **41**(1), 23–50 (2011)