Heuristic Anticipation Scheduling in Grid with Non-dedicated Resources

Victor V. Toporkov, Dmitry M. Yemelyanov(✉), and Petr A. Potekhin

National Research University "Moscow Power Engineering Institute", Moscow, Russia
{ToporkovVV,YemelyanovDM,PotekhinPA}@mpei.ru

Abstract. A heuristic user job-flow scheduling approach to grid virtual organizations with non-dedicated resources is discussed in this article. Users' and resource providers' preferences, virtual organization's internal policies, resources geographical distribution along with local private utilization impose specific requirements for efficient scheduling according to different, usually contradictive, criteria. The available resources set and the corresponding decision space decrease as resources utilization increases. This introduces further complications into the task of efficient scheduling. We propose a heuristic anticipation scheduling approach to improve the overall scheduling efficiency. Initially, it generates a near optimal but infeasible scheduling solution which is then used as a reference for efficient allocation of resources.

Keywords: Scheduling · Grid · Resources · Utilization · Heuristic · Job batch · Virtual organization · Anticipation

1 Introduction and Related Works

In distributed environments with non-dedicated resources, such as utility grids, 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 of slots, i.e. time intervals during which the individual computational nodes are capable of executing parts of independent users' parallel jobs. These slots generally have different start and finish times and present a difference in performance. The presence of a set of slots deprives the problem of a coordinated selection of the resources that are necessary to execute the job flow coming from computational environment users. Resource fragmentation also results in a decrease of the total computing environment utilization level [1,2].

Two established trends may be outlined among diverse approaches to distributed computing. The first one is based on the available resources utilization

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), the Ministry of Education and Science of the Russian Federation (project no. 2.9606.2017/BCh).

L. Sokolinsky and M. Zymbler (Eds.): PCT 2017, CCIS 753, pp. 58–70, 2017.
DOI: 10.1007/978-3-319-67035-5_5

and application level scheduling [3]. As a rule, this approach does not imply any global resource sharing or allocation policy. Another trend is related to the formation of user's virtual organizations (VO) and a job flow scheduling [4,5]. In this case, a metascheduler is an intermediate chain between the users, and local resource management and job batch processing systems.

Uniform rules of resource sharing and consumption, in particular based on economic models, make it possible to improve the job-flow level scheduling and resource distribution efficiency. VO policy may offer optimized scheduling to satisfy both users' and VO common preferences. 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], and so on.

VO formation and performance largely depends on mutually beneficial collaboration between all the related stakeholders. However, users' preferences and VO common preferences (owners' and administrators' combined) 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 available resources load balancing or node owners' profit boosting. Thus, VO policies in general should respect all members, and the most important aspect of the rules suggested by VO is their fairness.

A number of works understand fairness as it is defined in the theory of cooperative games, such as fair job flow distribution [8], fair quotas [13,14], fair user jobs prioritization [10], non-monetary distribution [15]. The cyclic scheduling scheme (CSS) [16] implements a fair scheduling optimization mechanism that ensures stakeholders interests to some predefined extent.

The downside of a majority of centralized metascheduling approaches is that they loose their efficiency and optimization features in distributed environments with a limited resource supply. 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. The common root cause is that, in fact, the same scarce set of resources (being efficient or not) has to be used for a job-flow execution, otherwise some jobs might hang in the queue. And under such conditions, user jobs priority and ordering greatly influence the scheduling results. At the same time, application-level brokers are still able to ensure user preferences and optimize the job's performance under free-market mechanisms.

A main contribution of this paper is a heuristic CSS-based job-flow scheduling approach that retains optimization features and efficiency even in distributed computing environments with limited resources. 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 a simulation experiment setup and results for the proposed scheduling approach. Finally, Sect. 5 summarizes the paper.

2 Cyclic Alternative-Based Fair Scheduling Model and Limited Resources

Scheduling of a job flow using the CSS is performed in time cycles known as scheduling intervals, by job batches [16]. 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 [17]. During the second step, dynamic programming methods [16] are used to choose an optimal alternatives combination (one alternative is selected for each job) with respect to the given VO and user criteria. This combination represents the final schedule based on current data regarding resources load and possible alternative executions.

An example of a user scheduling criterion may be a minimization of overall job running time, a minimization of overall running cost, etc. This criterion describes user's preferences for that specific job execution, and expresses a type of additional optimization to perform while searching for alternatives. Alongside with time (T) and cost (C) properties, each job execution alternative has a user utility (U) value: a 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 a Pareto-optimal strategy search involving both kinds of properties [4,16,18].

We consider the following relative approach to represent a user utility U. A job alternative with the minimum (the 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 of the user criterion, U is defined, depending on its position in the interval $[Z_{\min}; Z_{\max}]$, according to the following formula:

$$U = \frac{Z - Z_{\min}}{Z_{\max} - Z_{\min}} \cdot 100\%. \tag{1}$$

Thus, each alternative gets its utility in relation to the "best" and the "worst" optimization criterion values that a user could expect according to the job's priority. And the more some alternative corresponds to user's preferences, the smaller is the value of U. Examples of user utility functions for a job with four alternatives and a cost minimization criterion are presented in Table 1.

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

The launch of any job requires a co-allocation of a specified number of slots, in the same manner as in the classic backfilling variation. A single slot is a time span that can be assigned to run a part of a multiprocessor job. The target is to scan a list of N_s available slots, and to select a *window* of m parallel slots with the length of the required resource reservation time (see Fig. 1). The user job

Table 1. User utility examples for a job with execution cost minimization

Job execution alternatives	Execution cost	Utility
First alternative	5	0%
Second alternative	7	20%
Third alternative	11	60%
Fourth alternative	15	100%

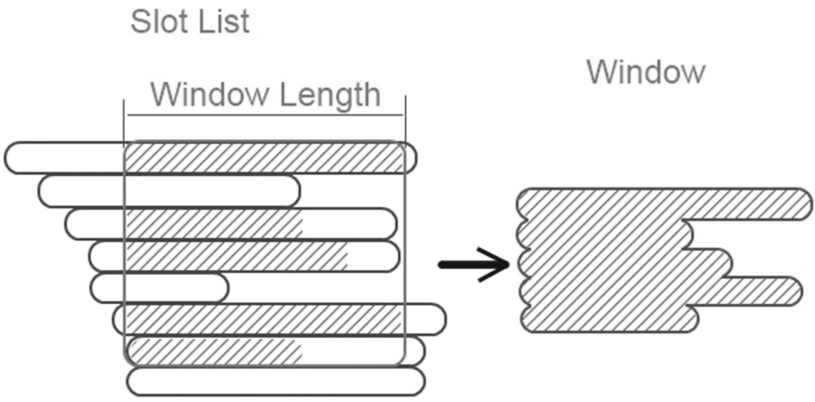

Fig. 1. An example of a window allocation procedure

requirements are arranged into a resource request containing a resource reservation time, characteristics of computational nodes (clock speed, RAM amount, disk space, operating system, etc.), and limitations on the selected window maximum cost. ALP, AMP and AEP window search algorithms were discussed in [17].

The job batch scheduling requires 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 overutilized, 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 [19] assign any job to the first set of slots matching the resource request conditions, without any optimization.

3 Heuristic Anticipation Scheduling

3.1 General Anticipation Scheduling Scheme

In order to address this problem, the following heuristic job batch scheduling scheme, consisting of three main steps, is proposed.

1. First, a set of all possible execution alternatives is found for each job, without considering time slots intersections or any resource reservation. The resulting intersecting alternatives found for each job reflect a full range of different job execution possibilities that a user may expect on the current scheduling interval. It may be noticed that this set is guaranteed to include the best and the worst alternatives according to any scheduling criterion, including user and VO criteria.
2. Second, the CSS procedure is performed to select alternatives combination (one alternative for each job of the batch) optimal according to the VO policy. The resulting alternatives combination most likely corresponds to an infeasible scheduling solution, since possible time-slots intersections will cause collisions in the resources allocation stage.
   The main idea of this step is that the obtained infeasible solution will provide some heuristic insights on how each job should be handled during the scheduling. For example, whether time-biased or cost-biased execution is preferred, how it should correspond to user criterion and VO administration policy, and so on.
3. Third, a feasible resources allocation is performed by replicating alternatives selected in step 2. The base for this replication step is an Algorithm searching for Extreme Performance (AEP) described in details in [17]. In the current step, AEP helps to find and reserve feasible execution alternatives most similar to those selected in the near-optimal infeasible solution.

After these three steps are performed, the resulting solution is both feasible and efficient, as it reflects a scheduling pattern obtained from a near-optimal reference solution from step 2.

The following subsections will discuss these scheduling steps in more details.

3.2 Finding a Near Optimal Infeasible Scheduling Solution

CSS results strongly depend on the diversity of alternatives sets obtained for batch jobs. The task of finding all possible execution alternatives for each job of the batch may become impractical, since the number of different resources combinations may reach $C(p, m)$, where p is the total number of different resource types available, and m is the number of resources requested by the user. Moreover, if we consider non-dedicated resources, then this task will be additionally complicated by local resources utilization. In this case, not all the resources combinations may be available during the scheduling cycle.

However, as we need to find alternatives for an *a priori* infeasible reference solution, a reasonable diverse set of possible execution alternatives will do. An important feature of this set is that it should contain extreme execution alternatives according to different criteria, e.g. the most expensive, the least time-consuming alternative, and so on.

Further, this set of possible alternatives may be used to evaluate actual user job execution against the job execution possibilities according to Eq. (1). We assume that such a set may represent a fair uniform basis for a user utility

expectations. This *uniform user utility* U_U may be used to compare different scheduling algorithms from the user's point of view.

We used a modification of the AEP to allocate a diverse set of execution alternatives for each job. Originally, the AEP is able to find only one alternative execution that satisfies the user resource request and is optimal according to the user custom criterion. The main idea of the current modification is to save all intermediate AEP search results to a dedicated list as shown in the following algorithm description:

```
Data: slotList — a list of available slots; job — a job for which the search is
      performed
Result: alternativesSet — a set of possible alternatives
slotList = orderSystemSlotsByStartTime();
for each slot in slotList do
    if not(properHardwareAndSoftware(slot.node)) then
        continue;
    end
    windowSlotList.add(slot);
    windowStartTime = slot.startTime;
    for each wSlot in windowSlotList do
        minLength = wSlot.node.getWorkingTimeEstimate();
        if (wSlot.endTime - windowStartTime) < minLength then
            windowSlotList.remove(wSlot);
        end
    end
    if windowSlotList.size() ≥ job.nodesNeed then
        minCostWindow = getMinCostWindow(windowSlotList);
        maxCostWindow = getMacCostWindow(windowSlotList);
        minRuntimeWindow =
            getMinRuntimeWindow(windowSlotList);
        alternativesSet.add(minCostWindow);
        alternativesSet.add(maxCostWindow);
        alternativesSet.add(minRuntimeWindow);
    end
end
```

Algorithm 1. AEP modification to allocate a set of possible job execution alternatives

In this algorithm, an expanded window *windowSlotList* of size M *moves* through a list *slotList* of all available slots sorted by their start time in ascending order. At each step, any combination of m slots inside *windowSlotList* (in the case when $m \leq M$) can form a window that meets all the requirements to run the job. The main difference from the original AEP algorithm is indicated in bold. Instead of searching for a single window with a maximum criterion value, we allocate several windows with extreme criteria values from every instance

of *windowSlotList*, and save to *alternativesSet*. By the end of *slotList*, *alternativesSet* will contain a diverse set of possible job execution alternatives. And since every possible *windowSlotList* instance is processed by the AEP, *alternativesSet* is guaranteed to contain alternatives with extreme criteria values (maxCost/minCost/minTime), as well as a variety of alternatives with some intermediate criteria values.

After sets of possible intersecting execution alternatives are allocated for each job, a CSS scheduling optimization procedure selects an optimal alternatives combination according to VO and users criteria [16].

3.3 Replication Scheduling and Resources Allocation

The resulting near-optimal scheduling solution in most cases is infeasible, since the selected alternatives may share the same time slots, thereby causing resource collisions. However, we suggest to use it as a reference solution, and replicate it into a feasible resources allocation.

For the purpose of replication, a new *Execution Similarity* criterion was introduced, which assists AEP in finding a window with 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 the reference alternative has total cost C_{ref}, 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, then the *distance* D between two alternatives may be calculated as a linear sum of criteria errors,

$$D_m = E_C + E_T + .. + E_U,$$

or as a geometric distance in a parameters space,

$$D_g = \sqrt{E_C^2 + E_T^2 + ... + E_U^2}.$$

For a feasible job batch resources allocation, the AEP consequentially allocates for each job a single execution window with a minimum *distance* to a reference alternative. Time slots allocated to the i-th job are reserved and excluded from the slot list when the AEP search algorithm is performed for the following jobs $i + 1, i + 2, ...N$. Thus, this procedure prevents any conflicts between resources and provides a scheduling solution that in some sense reflects a near-optimal reference solution.

4 Simulation Study

4.1 Simulation Environment Setup

An experiment was prepared as follows, using a custom distributed environment simulator [20]. Virtual organization and computing environment properties:

– The resource pool includes 80 heterogeneous computational nodes.
– The 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 the 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% of time quanta excluded in total.

Job batch properties:

– The number of jobs in a batch is 125.
– The number of nodes needed for a job is a whole number distributed evenly in $[2; 6]$.
– The node reservation time is a whole number distributed evenly in $[100; 500]$.
– The job budget varies in a way that some of the jobs can pay as much as 160% of the base cost, whereas some may require a discount.
– Every request contains a specification of a custom user criterion, namely one of the following: job execution runtime or overall execution cost.

During each experiment, a VO domain and a job batch were generated, and the following scheduling schemes were simulated and studied.

First, a general *CSS* solved the optimization problems $T \to \min, \lim U$ with different limits $U_a \in \{0\%, 1\%, 4\%, 10\%, 16\%, 32\%, 100\%\}$. U_a stands for the average user utility for one job, e.g. $\lim U_a = 10\%$ means that at average the resulting deviation from the best possible outcome for each user did not exceed 10%.

Second, a near-optimal but infeasible reference solution *REF* (see Sect. 3.2) was obtained for the same problems.

Third, a replication procedure CSS_{rep} was performed based on the *CSS* solution to demonstrate the replication process accuracy.

For the heuristic anticipation scheduling *ANT*, the same replication procedure was performed based on the *REF* solution.

Finally, two independent job batch scheduling procedures were performed to find the scheduling solutions most suitable for VO users ($USER_{\text{opt}}$) and VO administrators (VO_{opt}). $USER_{\text{opt}}$ was obtained applying only user criteria to allocate resources for jobs without taking into account VO preferences. VO_{opt} was obtained by using one VO optimization criterion (the runtime minimization $T \to \min$ in our example) for each job scheduling without taking into account user preferences.

4.2 Simulation Results

1000 single scheduling experiments were simulated. The average number of alternatives found for a job in *CSS* was 2.6. This result shows that usually a few alternative executions were found for relatively *small* jobs, whereas *large* jobs usually had at most one possible execution option (remember that according to the simulation settings the difference between jobs execution time could be up to 15-fold). At the same time, the *REF* algorithm at average considered more

than 100 alternative executions for each job. *CSS* failed to find any alternative execution at least for one job of the batch in 209 experiments; *ANT* did the same in 155 experiments.

These results show that the simulation settings provided a quite diverse job batch and, at the same time, a limited set of resources not allowing to execute all the jobs during every experiment.

Figure 2 shows the average job execution time (VO criterion) in a $T \rightarrow \min$, $\lim U$ optimization problem. Different limits $U_a \in \{0\%, 1\%, 4\%, 10\%, 16\%, 32\%, 100\%\}$ specify to what extent user preferences were taken into account. The two horizontal lines $USER_{\text{opt}}$ and VO_{opt} indicate, respectively, the practical T values when only user or VO administration criteria are optimized.

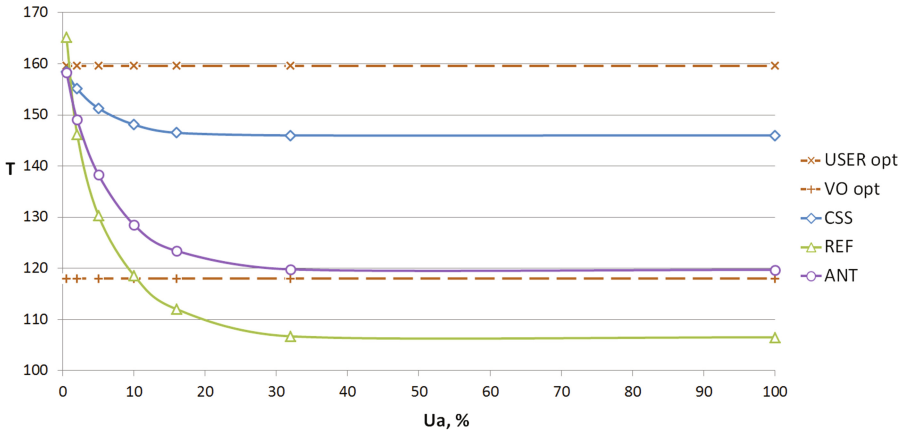

Fig. 2. Average job execution time in $T \rightarrow \min$, $\lim U$ problem

The first thing that attracts our attention in Fig. 2 is that *REF* provides for $U > 10\%$ a better (smaller) job execution time value than those of VO_{opt}. This behavior is, nonetheless, expected, as *REF* generates an infeasible solution and may use time-slots from more suitable (according to VO preferences) resources several times for different jobs.

On the other hand, *ANT* provided a better VO criterion value than *CSS* for all $U > 0\%$. The relative advantage reaches 20% when $U > 20\%$.

Interestingly, the *ANT* algorithm graph gradually changes from the $USER_{\text{opt}}$ value at $U = 0\%$ to almost the VO_{opt} value at $U = 100\%$ just as the average user utility limit changes. Therefore, *ANT* represents a general scheduling approach allowing to balance between VO stakeholders criteria according to a specified scenario, including VO or user criteria optimization.

A similar pattern can be observed in Fig. 3, where the $C \rightarrow \max$, $\lim U$ scheduling problem is represented. In this scenario, however, the *ANT* advantage over *CSS* amounts to 10% against the VO criterion.

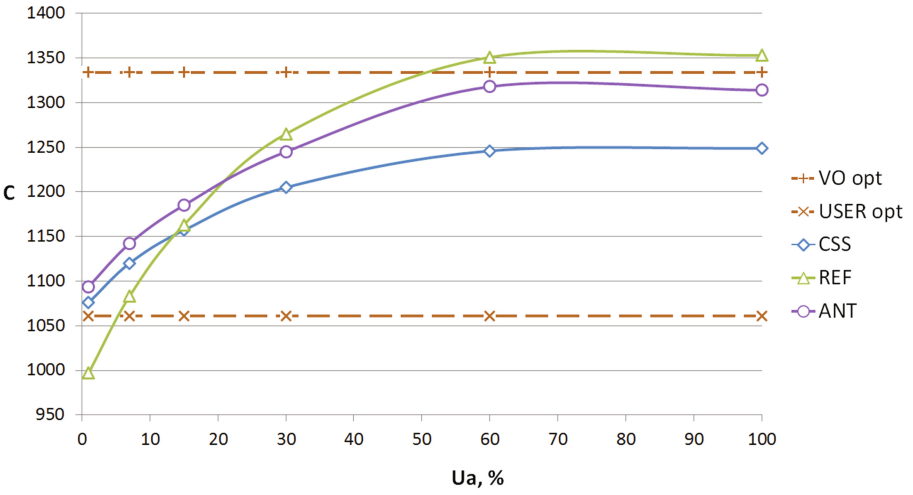

Fig. 3. Average job execution cost in the $C \rightarrow \max, \lim U$ problem

The advantage of *ANT* over *CSS* can be explained by a scarce set of alternatives found for user jobs by the latter algorithm. To compare *ANT* and *CSS* scheduling results against user criteria we used U_U uniform user utility metric (see Sect. 3.2). Figure 4 shows average uniform user utility U_U for *ANT* and *CSS* recalculated based on reference alternatives from *REF* using Eq. 1.

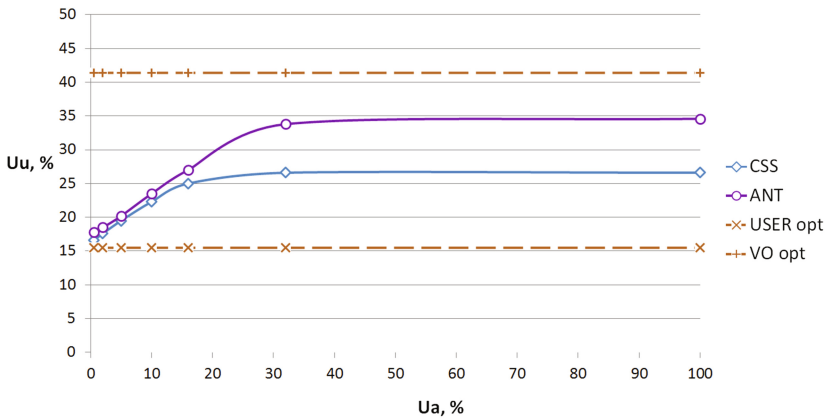

Fig. 4. Uniform user utility value in the $T \rightarrow \min, \lim U$ problem

As it can be seen from Fig. 4, *ANT* provides a higher uniform user utility in each experiment compared with *CSS*. Even more, *ANT* generally operates in a wider range of possible user utilities: from 17% when $U = 0\%$ (only user criteria

are optimized) to 35% when $U = 100\%$ (only the VO criterion is optimized). At the same time, *CSS* is able to change uniform user utility in the interval $(16\%; 26\%)$. Wider U_U interval represents greater decision space for *ANT* which implies optimization advantage.

It may seem that both algorithms operate in a rather small uniform user utilities interval and are not efficient enough. However, uniform user utilities are based on an infeasible set of alternatives and cannot be entirely replicated in a feasible solution. In Fig. 4, the VO_{opt} and $USER_{\text{opt}}$ horizontal lines roughly represent a feasible interval for uniform user utility values. In this case, *ANT* covers more than half of this feasible U_U interval.

Finally, Fig. 5 shows the average replication error (or *distance*) for each job of the batch provided by replicating a feasible *CSS* solution for $U_a = 30\%$ ($CSS_{\text{ref}}30$) and infeasible *REF* solutions with $\lim U_a = 0\%$ (*ANT*) and $\lim U_a = 30\%$ (*ANT*30). In this experiment, we used time and cost errors to calculate a geometric distance *Dg* between reference and allocated alternatives. Figure 5 shows that the CSS_{ref} error is practically independent from ordinal job number, reaching 0.05 (or 5%) for the last job of the batch. Thus, we can conclude that a feasible solution generally may be replicated with a good accuracy even when resources are limited.

Completely different results are provided by *ANT*. Depending on U_a, the *ANT* error may reach 0.35 (or 35%) for the last jobs of the batch. Unlike *CSS*, an infeasible *REF* solution may require, for example, the allocation of the nodes with the highest possible performance for each job. In this case, the replication process will not be able to reserve the required amount of high performance nodes, and the error for the last jobs may increase greatly.

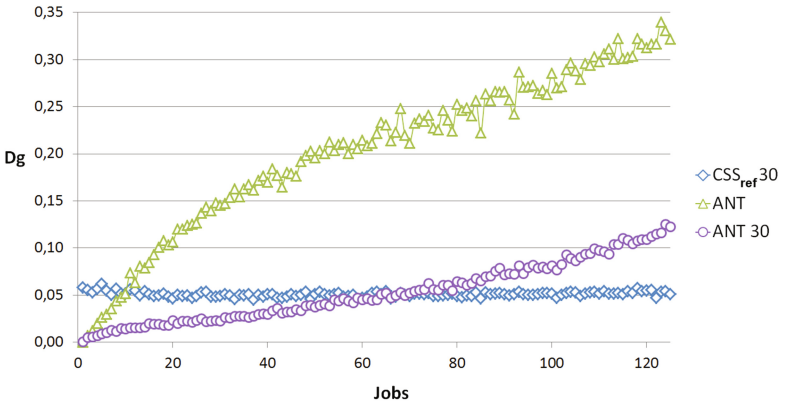

Fig. 5. Average replication error for user jobs

5 Conclusions and Future Work

In this paper, we have studied the problem of a fair job batch scheduling with a relatively limited resources supply. The main problem arising is a scarce set of job execution alternatives, which eliminates the scheduling optimization efficiency. We proposed an algorithm to obtain a diverse set of possible execution alternatives for each job. This set may be used as a uniform basis for a fair *uniform user utility* calculation to rate a scheduling solution. Then we proposed a heuristic scheduling scheme that generates a near-optimal but infeasible reference solution and, after that, replicates it to allocate a feasible accessible solution.

A computer simulation was performed to study these algorithms and evaluate their efficiency. The obtained results show that the new heuristic approach provides flexible and efficient solutions for different fair scheduling scenarios. The advantage over the general CSS against VO preferences (for example, when minimizing the total job batch execution time) reaches 25%. The above-mentioned replication procedure showed a relatively high accuracy providing less than 5% error when replicating a batch of 125 user jobs.

Future work will focus on the study of the replication algorithm and its possible application to fulfill complex user preferences expressed in a resource request.

References

1. Dimitriadou, S.K., Karatza, H.D.: Job scheduling in a distributed system using backfilling with inaccurate runtime computations. In: Proceedings of 2010 International Conference on Complex, Intelligent and Software Intensive Systems, pp. 329–336 (2010). doi:10.1109/CISIS.2010.65
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). doi:10.1007/s12652-015-0274-y
3. Buyya, R., Abramson, D., Giddy, J.: Economic models for resource management and scheduling in grid computing. J. Concurr. Comput. **14**(5), 1507–1542 (2002). doi:10.1002/cpe.690
4. 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, vol. 64, pp. 271–293. Springer, Boston (2003). doi:10.1007/978-1-4615-0509-9_18
5. Rodero, I., Villegas, D., Bobro, N., Liu, Y., Fong, L., Sadjadi, S.M.: Enabling interoperability among grid meta-schedulers. J. Grid Comput. **11**(2), 311–336 (2013). doi:10.1007/s10723-013-9252-9
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). doi: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), pp. 343–350 (2007). doi:10.1109/ccgrid.2007.52

8. Penmatsa, S., Chronopoulos, A.T.: Cost minimization in utility computing systems. Concurr. Comput. Pract. Exp. **16**(1), 287–307 (2014). doi:10.1002/cpe.2984. Wiley
9. 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). doi:10.1016/j.future.2014.11.019
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, pp. 291–297. IEEE Computer Society (2007). doi:10.1109/grid.2007.4354145
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). doi: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: 15th International Workshop JSSP 2010, vol. 6253, pp. 16–34 (2010). doi:10.1007/978-3-642-16505-4_2
13. 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). doi:10.1109/ispdc.2007.16
14. 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, pp. 50–57 (2007). doi:10.1109/grid.2007.4354115
15. Skowron, P., Rzadca, K.: Non-monetary fair scheduling cooperative game theory approach. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2013), pp. 288–297 (2013). doi:10.1145/2486159.2486169
16. 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). doi:10.1007/978-3-319-39639-2_44
17. Toporkov, V., Toporkova, A., Tselishchev, A., Yemelyanov, D.: Slot selection algorithms in distributed computing. J. Supercomput. **69**(1), 53–60 (2014). doi:10.1007/s11227-014-1210-1
18. Farahabady, M.H., Lee, Y.C., Zomaya, A.Y.: Pareto-optimal cloud bursting. IEEE Trans. Parallel Distrib. Syst. **25**, 2670–2682 (2014). doi:10.1109/tpds.2013.218
19. Cafaro, M., Mirto, M., Aloisio, G.: Preference-based matchmaking of grid resources with CP-Nets. J. Grid Comput. **11**(2), 211–237 (2013). doi:10.1007/s10723-012-9235-2
20. Toporkov, V., Tselishchev, A., Yemelyanov, D., Bobchenkov, A.: Composite scheduling strategies in distributed computing with non-dedicated resources. Procedia Comput. Sci. **9**, 176–185 (2012). doi:10.1016/j.procs.2012.04.019