Anticipation Scheduling in Grid Virtual Organizations

Victor Toporkov(✉), Dmitry Yemelyanov, Vadim Loginov, and Petr Potekhin

National Research University "MPEI", ul. Krasnokazarmennaya, 14, Moscow 111250, Russia
{ToporkovVV, YemelyanovDM, LoginovVA, PotekhinPA}@mpei.ru

Abstract. In this work, a job-flow scheduling approach for Grid virtual organizations is proposed and studied. 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. With increasing resources utilization level the available resources set and corresponding decision space are reduced. This further complicates the task of efficient scheduling. In order to improve overall scheduling efficiency we propose a heuristic anticipation scheduling approach. It generates a near optimal but infeasible scheduling solution and includes special replication procedure for efficient and feasible resources allocation.

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

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 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 computational environment users. Resource fragmentation also results in a decrease of the total computing environment utilization level [1, 2].

Application level scheduling [3] is based on the available resources utilization and, as a rule, does not imply any global resource sharing or allocation policy. Job flow scheduling [4, 5] in user's virtual organizations (VO) suppose uniform rules of resource sharing and consumption, in particular based on economic models. This approach allows improving 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], etc.

VO formation and performance largely depends on mutually beneficial collaboration between all the related stakeholders. However, users' preferences and owners' and administrators' 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 available resources load balancing or node owners' profit boosting. Thus, VO policies in general should respect all members 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, such as fair job flow distribution [9], fair quotas [13, 14], fair user jobs prioritization [10], and non-monetary distribution [15]. The cyclic scheduling scheme (CSS) [16] implements a fair scheduling optimization mechanism which ensures stakeholders interests to some predefined extent. The downside of a majority centralized metascheduling approaches is that they lose their efficiency and optimization features in distributed environments with a limited resources 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 general root cause is that in fact the same scarce set of resources (being efficient or not) have to be used for a job flow execution or otherwise some jobs might hang in the queue. 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.

Main contribution of this paper is a heuristic CSS-based job-flow scheduling approach which retains efficiency even in distributed computing environments with limited resources. Special scheduling solution *replication* procedure is proposed and studied to ensure a feasible scheduling result. 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 simulation experiment setup and results for the proposed scheduling approach. Finally, Sect. 5 summarizes the paper.

2 Cyclic Alternative-Based Fair Scheduling

Scheduling of a job flow using 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 the 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. An example for a user scheduling criterion may be an overall job running 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 [4, 16, 18]. For a fair CSS scheduling model the second step 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); $U \rightarrow \min$, lim T (meet user's best interests, while ensuring some acceptable job flow execution time) and so on [16].

We consider the following relative approach to represent a 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 using the following formula: $U = \frac{Z - Z_{\min}}{Z_{\max} - Z_{\min}} * 100\%$. 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. And the more some alternative corresponds to user's preferences the smaller is the value of U. For a fair scheduling model the second step 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); $U \rightarrow \min$, lim T (meet user's best interests, while ensuring some acceptable job flow execution time) and so on [16].

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 N_s available slots and to select a *window* of m 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 [17]. 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 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 Anticipation Scheduling

In order to address this problem the following heuristic job batch scheduling scheme is proposed which 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 user may expect on the current scheduling interval. Second, CSS scheduling procedure [16] 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 solution will provide some heuristic insights on how each job should be handled during the scheduling. For example, is 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 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 scheduling pattern obtained from a near-optimal reference solution from step 2.

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 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 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 $E_C = \frac{|C_{\mathrm{ref}} - C_{\mathrm{can}}|}{C_{\mathrm{ref}}}$. If one need 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 *Execution Similarity* criterion is represented below.

```
Input Data: slotList - a list of available slots; 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

slotList = orderSystemSlotsByStartTime();
minDistance = MAX_VALUE;

for each slot in slotList do
          if not(properHardwareAndSoftware(job, 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
          distance = calculateDistance(windowSlotList, refAlternative);
          if distance < minDistance then
                    minDistance = distance;
                    closestWindow = windowSlotList;
                    end
          end
end
```

In this algorithm an expanded window *windowSlotList* of size *M moves* through a whole list of all available slots *slotList* sorted by their start time in ascending order. At each step any combination of *m* slots inside *windowSlotList* (in the case, when *m* *M*) 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.

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

4 Simulation Study

An experiment was prepared as follows using a custom distributed environment simulator [2, 16, 17]. VO and computing environment properties:

- The resource pool includes 80 heterogeneous computational nodes.
- 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.

Job batch properties:

- Jobs number in a batch is 125.
- 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, a general CSS was performed in each experiment for the following job-flow execution cost maximization problem $C \to \max$, $\lim U_a = 10\%$. 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 E_c and processor time usage error E_t to calculate distances D_l and D_g.

In order to evaluate the resulting difference in scheduling outcomes, we additionally performed CSS algorithm for $C \to \max$, $\lim U_a = 0\%$ (ensuring users' individual preferences only) and $C \to \max$, $\lim U_a = 100\%$ (ensuring VO preference, i.e. maximizing overall cost without taking into account users' criteria) problems. 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.

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,

Table 1. CSS replication average scheduling results

Job execution characteristic	C -> max, lim U_a = 0%	C-> max, lim U_a = 10%	Linear replication	Geometric replication	C -> max, lim U_a = 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

i.e. characteristics which were used for a replication distance calculation. For example, *borderline* problems $C \rightarrow$ max, lim $U_a = 0\%$ and $C \rightarrow$ max, lim $U_a = 0\%$ 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.

To address this discrepancy in more details Fig. 1 shows average linear and geometric replication distances for each job of the batch. Figure 1 shows that there values are practically independent from an ordinal job number and do not exceed 0.05. For comparison average distances between the most and the least expensive alternative executions for the first batch job amounted: $D_l = 1.15$ and $D_g = 0.88$. These values exceed average replication distances in 20 times and therefore are not shown in the Fig. 1. Thus, we can conclude that replication error for each batch job on average does not exceed 5% against interval of possible scheduling outcomes.

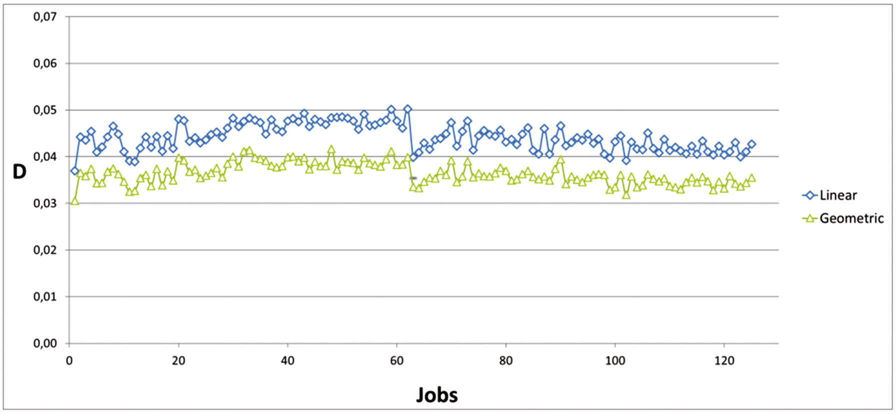

Fig. 1. Average replication error for user jobs

4.2 Anticipation Scheduling Simulation

The second experiment series consider anticipation scheduling efficiency. 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 $T \rightarrow \min$, lim U with different limits $U_a \in \{0\%, 1\%, 4\%, 10\%, 16\%, 32\%, 100\%\}$. Second, a near-optimal but infeasible reference solution REF was obtained for the same problems. Third, a replication procedure CSS_{rep} was performed based on CSS solution to demonstrate a replication process accuracy. For the heuristic anticipation scheduling ANT the same replication procedure was performed based on REF solution. We used a geometric distance as a replication criterion. Finally two independent job batch scheduling procedures were performed to find scheduling solutions most suitable for VO users (USER_{opt}) and VO administrators (VO_{opt}). USER_{opt} was obtained by using only user criteria to allocate resources for jobs without taking into account VO preferences. VO_{opt} was obtained by using one VO optimization criterion ($T \rightarrow \min$) for each job scheduling without taking into account user preferences.

1000 single scheduling experiments were simulated. Average number of alternatives found for a job in CSS was 2.6. This result shows that while for relatively *small* jobs usually a few alternative executions have been found, *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 times). At the same time REF algorithm at average considered more than 100 alternative executions for each job. CSS failed to find any alternative executions for at least for one job of the batch in 209 experiments; ANT - in 155 experiments. These results show that simulation settings at the same time provided quite a diverse job batch and a limited set of resources not allowing executing all the jobs during every experiment.

Figure 2 shows 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. Two horizontal lines USER_{opt} and VO_{opt} represent practical T values when only user or VO administration criteria are optimized correspondingly.

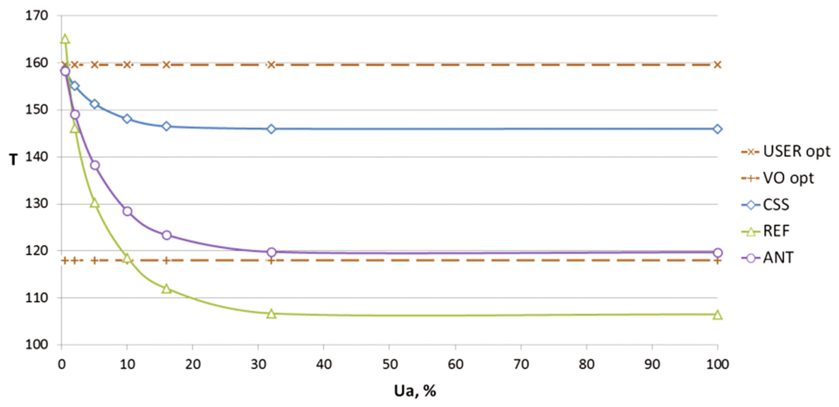

Fig. 2. Average job execution time in $T \rightarrow \min$, lim U problem

First thing that catches the eye in Fig. 2 is that REF for $U > 10\%$ provides job execution time value better (smaller) than those of VO_{opt}. However such behavior is 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. Otherwise ANT provided better VO criterion value than CSS for all $U > 0\%$. The relative advantage reaches 20% when $U > 20\%$ is considered. ANT algorithm graph gradually changes from $USER_{opt}$ value at $U = 0\%$ to almost VO_{opt} value at $U = 100\%$ just with changing average user utility limit. Thereby ANT represents a general scheduling approach allowing balancing between VO stakeholder's criteria according to specified scenario, including VO or user criteria optimization.

A similar pattern can be observed in Fig. 3 where $C \rightarrow \max$, lim U scheduling problem is presented. However, in this case ANT advantage over CSS amounts to 10% against VO criterion.

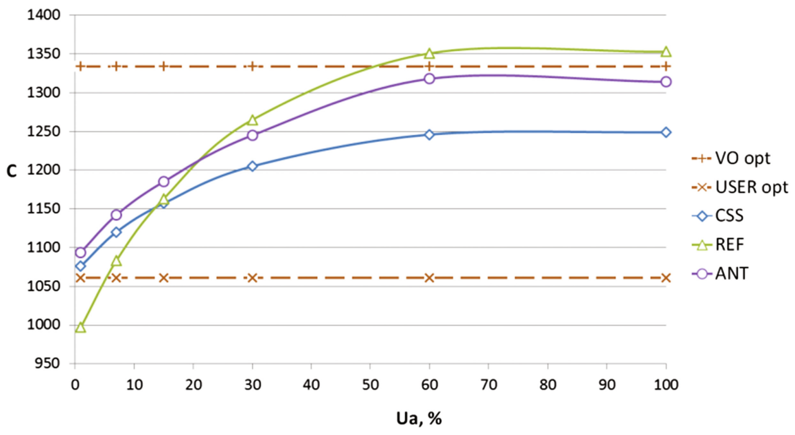

Fig. 3. Average job execution cost in $C \rightarrow \max$, lim U problem

5 Conclusions and Future Work

In this paper, we study the problem of a fair job batch scheduling with a relatively limited resources supply. The main problem arise is a scarce set of job execution alternatives which eliminates scheduling optimization efficiency. We study a heuristic scheduling scheme which generates a near-optimal but infeasible reference solution and then replicates it to allocate a feasible accessible solution. 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.

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

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/BCh).

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. Buyya, R., Abramson, D., Giddy, J.: Economic models for resource management and scheduling in grid computing. J. Concurrency Comput. **14**(5), 1507–1542 (2002)
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, pp. 271–293. Kluwer Academic Publishers (2003)
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., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP, vol. 18, pp. 128–152. Springer, Heidelberg (2002)
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., Lrida, J.L., Albornoz, V.M.: MIP model scheduling for multi-clusters. In: Euro-Par 2012, pp. 196–206. Springer, Heidelberg (2012)
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: Schwiegelshohn, U., Frachtenberg, E., (eds.) JSSPP 2010, vol. 6253, pp. 16–34. Springer, Heidelberg (2010)
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)
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, Austin, USA, pp. 50–57. IEEE Computer Society (2007)

15. 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)
16. Toporkov, V., Yemelyanov, D., Bobchenkov, A., Tselishchev A.: Scheduling in grid based on VO stakeholders preferences and criteria. Advances in Intelligent Systems and Computing, vol. 470, pp. 505–515. Springer International Publishing Switzerland (2016)
17. Toporkov, V., Toporkova, A., Tselishchev, A., Yemelyanov, D.: Slot selection algorithms in distributed computing. J. Supercomput. **69**(1), 53–60 (2014)
18. Farahabady, M.H., Lee, Y.C., Zomaya, A.Y.: Pareto-optimal cloud bursting. IEEE Trans. Parallel Distrib. Syst. **25**, 2670–2682 (2014)
19. Cafaro, M., Mirto, M., Aloisio, G.: Preference-based matchmaking of grid resources with CP-Nets. J. Grid Comput. **11**(2), 211–237 (2013)