J Supercomput (2014) 69:53–60
DOI 10.1007/s11227-014-1210-1

Slot selection algorithms in distributed computing

Victor Toporkov · Anna Toporkova · Alexey Tselishchev · Dmitry Yemelyanov

Published online: 18 May 2014

Abstract In this work, we introduce slot selection and co-allocation algorithms for parallel jobs in distributed computing with non-dedicated and heterogeneous resources. A single slot is a time span that can be assigned to a task, which is a part of a job. The job launch requires a co-allocation of a specified number of slots starting synchronously. The challenge is that slots associated with different resources of distributed computational environments may have arbitrary start and finish points that do not match. Some existing algorithms assign a job to the first set of slots matching the resource request without any optimization (the first fit type), while other algorithms are based on an exhaustive search. In this paper, algorithms for effective slot selection of linear complexity on an available slots number are studied and compared with known approaches. The novelty of the proposed approach consists of allocating alternative sets of slots. It provides possibilities to optimize job scheduling.

Keywords Distributed computing · Economic scheduling · Resource management · Slot · Job · Task · Batch

V. Toporkov (✉) · D. Yemelyanov
National Research University “MPEI”, ul. Krasnokazarmennaya, 14, Moscow 111250, Russia
e-mail: ToporkovVV@mpei.ru

D. Yemelyanov
e-mail: YemelyanovDM@mpei.ru

A. Toporkova
National Research University Higher School of Economics, Moscow State Institute of Electronics and Mathematics, Bolshoy Trekhsvyatitelsky per., 1-3/12, Moscow 109028, Russia
e-mail: AToporkova@hse.ru

A. Tselishchev
European Organization for Nuclear Research (CERN), 1211 Geneva 23, Switzerland
e-mail: Alexey.Tselishchev@cern.ch

1 Introduction

Economic mechanisms are used to solve problems like resource management and scheduling of jobs in a transparent and efficient way in distributed environments such as cloud services and utility Grid [1,2]. A resource broker model is decentralized, well scalable and application-specific [2,3]. The simultaneous satisfaction of various application optimization criteria submitted by independent users is not possible due to several reasons [2] and also can deteriorate such quality of service rates as total execution time of a sequence of jobs or overall resource utilization. Another model is related to virtual organizations (VO) [4–6] and metascheduling with central schedulers or a metabroker [2] providing job-flow level scheduling and optimization. VOs naturally restrict the scalability, but uniform rules for allocation and consumption of resources make it possible to improve the efficiency of resource usage and to find a trade-off between contradictory interests of different participants. The metaschedulers implement the economic policy of a VO based on local resource schedules. The schedules are defined as sets of slots coming from resource managers or schedulers in the resource domains. During each scheduling cycle, the sets of available slots are updated based on the information from local resource managers. Thus, during every cycle of the job batch scheduling [5], two problems have to be solved: (1) selecting an alternative set of slots (alternatives) that meet the requirements (resource, time and cost); (2) choosing a slot combination that would be the efficient or optimal in terms of the whole job batch execution in the current cycle of scheduling. To implement this scheduling scheme, first of all, one needs to propose the algorithm for finding sets of alternative executions. An optimization technique for the second phase of this scheduling scheme was proposed in [5,6].

The scheduling problem in Grid is NP-hard due to its combinatorial nature, and many heuristic-based solutions have been proposed. In [3], heuristic algorithms for slot selection, based on user-defined utility functions, are introduced. NWIRE system [3] performs a slot window allocation based on the user-defined efficiency criterion under the maximum total execution cost constraint. However, the optimization occurs only on the stage of the best found offer selection. First fit slot selection algorithms (backtrack [7] and NorduGrid [8] approaches) assign any job to the first set of slots matching the resource request conditions, while other algorithms use an exhaustive search [2,9,10] and some of them are based on a linear integer programming [2,9] or mixed-integer programming model [10]. The proposed in [2,9,10] scheduling techniques are effective compared with other scheduling techniques under given criteria: the minimum processing cost, the overall makespan, resources utilization etc. However, complexity of the scheduling process is extremely increased by the resources heterogeneity and the co-allocation process, which distributes the tasks of parallel jobs across resource domain boundaries. The degree of complexity may be an obstacle for online use in large-scale distributed environments. In our previous works [11–13], two algorithms for slot selection AMP and ALP that feature linear complexity $O(l)$, where l is the number of available time-slots, were proposed. Both algorithms perform the search of the first fitting window without any optimization. AMP (algorithm based on maximal job price), performing slot selection based on the maximum slot window cost, proved the advantage over ALP (algorithm based on local price of slots) when applied to the above-mentioned scheduling scheme. However, in order to accommodate an end users job execution requirements, there is a need for a more precise slot selection algorithm to exploit during the first stage of the proposed scheduling scheme and to consider various user demands along with the VO resource management policy. In this paper, we propose algorithms for effective slot selection based on user-defined criteria that feature linear complexity on the number of the available slots during the job batch scheduling cycle. The novelty of the proposed approaches consists of allocating a number of alternative sets of slots (alternatives). The proposed algorithms can be used for both homogeneous and heterogeneous resources.

The paper is organized as follows. Section 2 introduces a general scheme and its implementations for searching alternative slot sets that are effective by the specified criteria. Section 3 contains simulation results for comparison of proposed and known algorithms. Section 4 summarizes the paper and describes further research topics.

2 Algorithm searching for extreme performance

In this section, we consider a general scheme of an algorithm searching for extreme performance (AEP) and its implementation examples. The launch of any job requires a co-allocation of a specified number of slots, as well as in the classic backfilling variation (http://www.adaptivecomputing.com). The job resource 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.) and the limitation on the selected window maximum cost. According to the resource request, it is required to find a window with the following description: n concurrent time-slots providing the resource performance rate and the maximal resource price per time unit should be reserved for a required time span. The length of each slot in the window is determined by the performance rate of the node on which it is allocated. The window search is performed on the list of all available slots sorted by their start time in ascending order. The algorithm considers every combination of parallel slots that can form an "extended" window of $m \geq n$ slots. In addition, one can define a criterion crW on which the best matching window alternative is chosen: cost, execution runtime or, for example, energy consumption. The AEP scheme for an effective window search by the specified criterion crW is represented by Algorithm 1. A variable $bestWindow$ will contain an effective window by the given criterion crW. The algorithm parses a list of all available slots subsequently for all the batch jobs. Higher priority jobs are processed first [5].

The need to choose alternative sets of slots for every batch job increases the complexity of the whole scheduling scheme [5]. With a large number of the available slots, the algorithm execution time may become inadequate. Though it is possible to mention some typical optimization problems, based on the AEP scheme that can be solved with a relatively decreased complexity. These include problems of total job cost, runtime minimizing, the window formation with the minimal start/finish time. Let us consider as an example the procedure for *minimizing a window start time*. This procedure can be reduced to finding a set of the first n parallel slots the total cost of which does not exceed the budget limit. This description coincides the *AMP* scheme considered in previous works [11–13].

Algorithm 1: AEP scheme for an optimal slot window selection

```
Data: slotList - an list of available slots ordered by the start time; job
      - a job for which the search is performed
Result: bestWindow - a window with the extreme criterion crW value
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
        curWindow = getBestWindow(windowSlotList);
        crW = getCriteriaValue(curWindow);
        if crW > maxCriteriaValue then
            maxCriteriaValue = crW;
            bestWindow = curWindow;
        end
    end
end
```

For the proposed AEP efficiency analysis, the following implementations were added to the simulation model [5,6]: (1) *AMP* for searching alternatives with the earliest start time; (2) *MinFinish* for searching alternatives with the earliest finish time; (3) *MinCost* for searching a single alternative with the minimum total allocation cost; (4) *MinRunTime*: This algorithm performs a search for a single alternative with the minimum execution runtime; (5) *Common Stats, AMP* (further referred to as *CSA*) for searching multiple alternatives using *AMP*. To compare the search results with the algorithms, presented above, only alternatives with the extreme value of the given criterion are selected, so the optimization will take place at the selection process. Considered optimization includes job's start time, finish time, total execution cost and runtime minimization. It is worth mentioning that all proposed AEP implementations have a linear complexity $O(l)$: algorithms "move" through the list of all the available slots in the direction of non-decreasing start time without turning back or reviewing previous steps.

3 Experimental studies of slot selection algorithms

The goal of the experiment is to examine AEP implementations: to analyze alternatives search results with different efficiency criteria, to compare the results with *AMP* and to estimate the possibility of using in real systems considering the algorithm executions time. A simulation framework [5,6] was configured in a special way in order to study and to analyze the algorithms presented. In each experiment, a generation of the distributed environment that consists of 100 CPU nodes was performed. The relatively

Table 1 Proposed algorithms scheduling results.

Slot selection algorithm	Start time	Runtime	Finish time	Processor time	Cost
CSA	0	38	52.6	168.6	1352
AMP	0	100.2	100.2	308.6	1445.2
MinRunTime	53	33	86.1	158	1464.9
MinFinishTime	0	34.4	34.4	161.9	1464.2
MinCost	193	114.7	307.7	344.5	1027.3

high number of the generated nodes has been chosen to allow *CSA* to find more slot alternatives. Therefore, more effective alternatives could be selected for the searching results comparison based on the given criteria. The performance rate for each node was generated as a random integer variable in the interval [2; 10] with a uniform distribution. The resource usage cost was formed proportionally to their performance with an element of normally distributed deviation in order to simulate a free market pricing model [1–3]. The level of the resource initial load with the local and high priority jobs at the scheduling interval [0; 600] was generated by the hyper-geometric distribution in the range from 10 to 50 % for each CPU node. Based on the generated environment, the algorithms performed the search for a single initial job that required an allocation of five parallel slots for 150 Units of time. The maximum total execution cost according to user requirements was set to 1,500.

Experimental results are presented in Table 1. Let us consider the *average start time* for the alternatives found (and selected) by the aforementioned algorithms. *AMP*, *MinFinish* and *CSA* were able to provide the earliest job start time at the beginning of the scheduling interval ($t = 0$). The result was expected for *AMP* and *CSA* (which is essentially based on the multiple runs of the *AMP* procedure) since 100 available resource nodes provide a high probability that there will be at least five parallel slots starting at the beginning of the interval and can form a suitable window. The fact that the *MinFinish* algorithm was able to provide the same start time can be explained by the local tasks minimum length value, that is equal to 10. Indeed, the window start time at the moment $t = 10$ cannot provide the earliest finish time even with use of the most productive resources (for example the same resources allocated for the window with the minimal runtime).

The *minimum execution runtime* was obviously provided by the *MinRunTime* algorithm. Though, schemes *MinFinish* and *CSA* provide quite comparable values. High result for the *MinFinish* algorithm can be explained by the “need” to complete the job as soon as possible with the minimum (and usually initial) start time. Algorithms *MinFinish* and *MinRunTime* are based on the same criterion selection procedure. However, due to non-guaranteed availability of the most productive resources at the beginning of the scheduling interval, *MinRunTime* has the advantage. Relatively long runtime was provided by *AMP* and *MinCost* algorithms. For *AMP*, this is explained by the selection of the first fitting (and not always effective by the given criterion) alternative, while *MinCost* tries to use relatively cheap and (usually) less productive CPU nodes.

Table 2 Algorithms working time (in ms) depending on the processor nodes number

CPU nodes number	50	100	200	300	400
AMP	0.3	0.5	1.1	1.6	2.2
MinRunTime	3.2	12	45.5	97.2	169.2
MinFinishTime	3.2	12	45.1	96.9	169
MinCost	1.7	6.3	23.6	52.3	91.5

The *minimum average finish time* was provided by the *MinFinish* algorithm. *CSA* has the closest resulting finish time. The relative closeness of these values comes from the fact that other related algorithms did not intend to minimize a finish time value and were selecting windows without taking it into account. The late average finish time is provided by the *MinCost* scheme. This value can be explained not only with a relatively late average start time, but also with a longer (compared to other approaches) execution runtime due to the use of less productive resource nodes. The minimum value of the used *processor time* (the sum length of the window slots) is provided by *MinRunTime*. *MinFinish* and *CSA* were able to provide comparable results. The most processor time-consuming alternatives were obtained by *AMP* and *MinCost* algorithms. Similarly to the execution runtime value, this can be explained by using a random first fitting window (in case of *AMP*) or by using less expensive, and hence less productive, resource nodes (in case of the *MinCost* algorithm), as nodes with a low performance level require more time to execute the job.

Finally, let us consider the total *job execution cost*. The *MinCost* algorithm has a big advantage over other algorithms presented: It was able to provide the total cost of 1,027.3 (note that the total cost limit was set by the user at 1,500). Thus, the cheapest alternatives found by *CSA* have the average total execution cost equal to 1,352, that is 31.6 % more expensive compared to the result of the *MinCost* scheme, while alternatives found by *MinRunTime* (the most expensive ones) are 42.5 % more expensive.

The important factor is a complexity and an actual working time of the algorithms, especially with the assumption of the algorithm's repeated use during the first stage of the scheduling scheme [5]. Table 2 shows the actual algorithm execution time in milliseconds measured depending on the number of CPU nodes. The simulation was performed on a regular PC workstation with Intel Core i3 (2 cores @ 2.93 GHz), 3GB RAM on JRE 1.6, and 1,000 separate experiments were simulated for each value of the processor nodes numbers 50, 100, 200, 300, 400.

The average working time of *MinRunTime* and *MinCost* proves their (at most) quadratic complexity on the number of CPU nodes, and the execution times are suitable for practical use. The *AMP*'s execution time shows even near linear complexity because with a relatively large number of free available resources it was usually able to find a window at the beginning of the scheduling interval.

Table 3 contains the algorithms working time in milliseconds measured depending on the scheduling interval length. Overall 1,000 single experiments were conducted for each value of the interval length 600, 1,200, 1,800, 2,400, 3,000, 3,600. When analyzing the presented values, it is easy to see that all proposed algorithms have a

Table 3 Algorithms working time (in ms) depending on the scheduling interval length

Scheduling interval length	600	1200	1800	2400	3000	3600
AMP	0.5	0.82	1.1	1.44	1.79	2.14
MinRunTime	11.7	26	40.9	55.5	69.4	84.6
MinFinishTime	11.6	25.7	40.6	55.3	69	84.1
MinCost	6.1	13.4	20.9	28.5	35.7	43.5

linear complexity with respect to the length of the scheduling interval and, hence, to the number of the available slots, and their executions times are suitable for online scheduling.

4 Conclusions and future work

In this work, we address the problem of slot selection and co-allocation for parallel jobs in distributed computing with non-dedicated resources. Each of the AEP algorithms possesses a linear complexity on a total available slots number and a quadratic complexity on a CPU nodes number. The advantage of AEP-based algorithms over the general *CSA* scheme was shown for each of the considered criteria: start time, finish time, runtime, CPU usage time and total cost.

As a result, it may be stated that each full AEP-based scheme is able to obtain the best result in accordance with the given criterion. This allows to use the proposed algorithms within the whole scheduling scheme [5] at the first stage of the batch job alternatives search. Besides, a single run of the AEP-like algorithm had an advantage of 10–50 % over suitable alternatives found with *AMP* with respect to the specified criterion. A directed alternative search at the first stage of the proposed scheduling approach [5,6] can affect the final distribution and may be favorable for the end users.

In our further work, we will refine resource co-allocation algorithms in order to integrate them with scalable co-scheduling strategies [5,6].

Acknowledgments This work was partially supported by the Council on Grants of the President of the Russian Federation for State Support of Leading Scientific Schools (SS-362.2014.9), the Russian Foundation for Basic Research (Grant No. 12-07-00042), and by the Federal Target Program Research and scientific-pedagogical cadres of innovative Russia(state contract no. 16.740.11.0516).

References

1. Lee YC, Wang C, Zomaya AY, Zhou BB (2012) Profit-driven Scheduling for Cloud Services with Data Access Awareness. J. Parallel and Distributed Computing 72(4):591–602
2. Garg SK, Konugurthi P, Buyya R A Linear Programming-driven Genetic Algorithm for Meta-scheduling on Utility Grids. J. Parallel, Emergent and Distributed Systems 26, 493–517 (2011)
3. Ernemann C, Hamscher V, Yahyapour R (2002) Economic Scheduling in Grid Computing. In: Feitelson DG, Rudolph L, Schwiegelshohn U (eds) JSSPP 2002, vol 2537, LNCSSpringer, Heidelberg, pp 128–152
4. Kurowski K, Nabrzyski J, Oleksiak A, Weglarz J (2003) Multicriteria Aspects of Grid Re-source Management. In: Nabrzyski J, Schopf JM, Weglarz J (eds) Grid resource management. Kluwer Acad. Publishers, State of the art and future trends, pp 271–293

5. Toporkov V, Tselishchev A, Yemelyanov D, Bobchenkov A (2012) Composite Scheduling Strategies in Distributed Computing with Non-dedicated Resources. Procedia Computer Science. Elsevier 9:176–185
6. Toporkov V, Tselishchev A, Yemelyanov D, Bobchenkov A (2012) Dependable Strategies for Job-flows Dispatching and Scheduling in Virtual Organizations of Distributed Computing Environments. Complex Systems and Dependability, vol 170, AISCSpringer, Heidelberg, pp 240–255
7. Aida K, Casanova H (2008) Scheduling Mixed-parallel Applications with Advance Reservations. 17th IEEE Int., Symposium on HPDCIEEE CS Press, New York, pp 65–74
8. Elmroth E, Tordsson J A Standards-based Grid Resource Brokering Service Supporting Advance Reservations, Coallocation and Cross-Grid Interoperability. J. Concurrency and Computation: Practice and Experience 25(18), pp 2298–2335 (2009)
9. Takefusa A, Nakada H, Kudoh T, Tanaka Y (2010) 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, vol 6253, LNCSSpringer, Heidelberg, pp 16–34
10. Blanco H, Guirado F, Lrida JL, Albornoz VM (2013) MIP Model Scheduling for Multi-clusters. Euro-Par 2012, vol 7640, LNCSSpringer, Heidelberg, pp 196–206
11. Toporkov V, Toporkova A, Bobchenkov A, Yemelyanov D (2011) Resource Selection Algorithms for Economic Scheduling in Distributed Systems. Procedia Computer Science. Elsevier 4:2267–2276
12. Toporkov V, Yemelyanov D, Toporkova A, Bobchenkov A (2011) Resource Co-allocation Algorithms for Job Batch Scheduling in Dependable Distributed Computing. Dependable Computer Systems, vol 97, AICSSpringer, Heidelberg, pp 243–256
13. Toporkov V, Bobchenkov A, Toporkova A, Tselishchev A, Yemelyanov D (2011) Slot Selection and Co-allocation for Economic Scheduling in Distributed Computing. In: Malyshkin V (ed) PaCT 2011, vol 6873, LNCSSpringer, Heidelberg, pp 368–383