Stanisław DENIZIAK, **Paweł PADUCH**
KIELCE UNIWESITY OF TECHNOLOGY, DEPARTMENT OF INFORMATION SYSTEMS
7 Tysiąclecia Państwa Polskiego Ave., 25-314 Kielce, Poland

The scheduling of task graphs in high performance computing as service clouds

Abstract

In this paper we propose a new method of scheduling the distributed applications in cloud environment according to the High Performance Computing as a Service concept. We assume that applications, that are submitted for execution, are specified as task graphs. Our method dynamically schedules all the tasks using resource sharing by the applications. The goal of scheduling is to minimize the cost of resource hiring and the execution time of all incoming applications. Experimental results showed that our method gives significantly better utilization of computational resources than existing management methods for clouds.

Keywords: high performance computing, cloud computing, scheduling, distributed processing, Internet services.

1. Introduction

The advent of cloud computing has provided new opportunities to deliver Internet services. In addition to supporting software, platforms and infrastructure services, the concept of High Performance Computing as a Service (HPCaaS) has been proposed [1]. According to this concept, cloud providers deliver computing resources to client applications on pay-per-use basis. HPCaaS is particularly attractive for scientific applications requiring high performance computing facilities, as an alternative to dedicated supercomputers [2]. But to take the advantage of clouds for HPCaaS, some effective methods for cloud management should be developed. Especially efficient scheduling strategies should be applied [3].

HPCaaS is a relatively new concept, but some dedicated methods of task scheduling have already been proposed. Most of them are based on classical job scheduling algorithms adopted to cloud environment. In [2], the application aware scheduling has been proposed. This method takes into consideration multi-platform feature of the cloud. CLOUDRB [4] is an environment for scheduling HPC applications in the cloud. It takes into consideration user-defined deadlines. ASETS [5] is a task scheduling based on the idea of software-defined networking. In [6], the moldable job scheduling method has been proposed.

Scheduling strategies for clouds take into consideration parallel processing in applications [7]. For such applications more than one processor is allocated during scheduling. But usually, as it is not possible to parallelize all computations, this leads to inefficiency in resource utilization. In [8], the method of static scheduling of applications specified as echo algorithms has been presented. It has been shown that efficient scheduling, which takes into consideration resource sharing, may reduce the cost of hiring resources by up to 75%.

In this paper, we propose another method for scheduling distributed applications in HPCaaS cloud. Applications are specified as distributed algorithms represented by task graphs. The goal of our method is to minimize the cost of resource hiring and the execution time of all incoming applications. Experimental results showed that our method gives significantly better utilization of computational resources than existing scheduling methods dedicated to HPCaaS.

The paper is organized as follows. In the next section the concept of HPCaaS cloud is outlined. Section 3 describes the assumptions concerning the system specification. In Section 4 our approach is presented. Finally, experimental results and conclusions are given in Sections 5 and 6, respectively.

2. HPCaaS Cloud

The HPCaaS cloud environment is presented in Fig. 1. First, a user should develop an application implementing required computations (1). Next, the application is submitted to the cloud for execution (2). The user may specify also additional requirements, e.g. required resources, priority, time constraints, information about tasks etc. The HPCaaS manager schedules all applications, according to the above requirements and the management policy (3). After finishing execution of the application, the results are sent to the user (4) and the user receives the final report (5) containing payment information.

HPCaaS cloud resources usually are provided to users using "pay per-use" cost model. Existing cloud management systems allocate required resources to the application and the user pays for hiring the resources during the whole execution time. Since allocation of resources is performed on application level, the cloud manager allocates dedicated resources to each application.

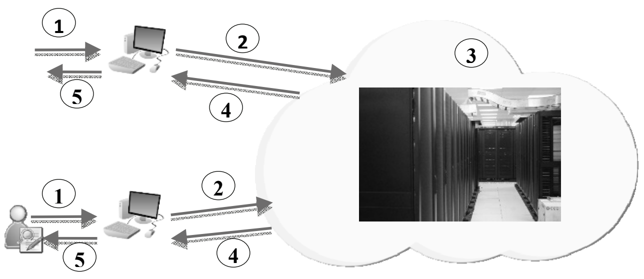

Fig. 1. HPCaaS environment

HPC applications usually require many processors to execute applications in a reasonable time. This requires to implement computations as a parallel or distributed application. Since it is difficult to achieve high utilization of all processors during parallel processing, the users pay for resources even when they are idle. To reduce the costs of HPCaaS services, we propose the method of sharing processors between different applications. Such an approach requires to schedule parallel applications on the task level as well as using the standardized, schedulable model of specification.

3. System specification

We assume that the target architecture for HPCaaS is a computing cluster composed of N identical nodes. We also assume that HPC applications are specified as task graphs. Data transmissions between tasks will not be considered i.e. communication is implemented using shared memory (e.g. disk subsystems) or transmission times are not significant and may be neglected.

The functional specification of HPC application is represented by an acyclic graph $G=\{T, E\}$, where T is a set of nodes corresponding to tasks and E is a set of edges. Edges exist only between nodes corresponding to communicating tasks. Any task may start its execution after finishing execution of all its predecessors. Loops are specified as separate task graphs of the multi-rate system. Sample specification of the task graph consisting of 12 processes is presented in Fig. 2.

We assume that the time of execution for each task is known. It may be estimated using the worst cease estimation. Table 2 presents estimations for the tasks from Fig. 1. Tasks 1 and 12 are source and sink nodes, they do not correspond to any computations, hence, their execution times are equal to 0. Such tasks are usually added to task graphs to define the entry and exit points for computations. These tasks will be omitted during scheduling. We assume that estimations of execution times are given by the user.

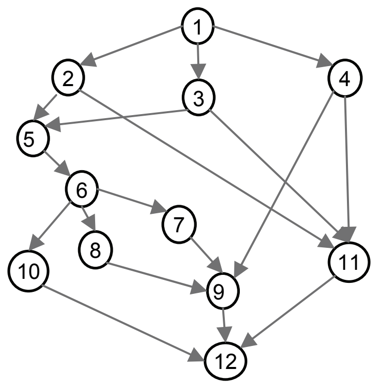

Fig. 2. Sample task graph

Tab. 1. Execution times of tasks from Fig. 1

Task	1	2	3	4	5	6	7	8	9	10	11	12
Time [s]	0	4	7	2	8	3	8	3	8	4	6	0

4. Task scheduling

Our method schedules applications in the same order they are submitted for execution. The aim of optimization is to obtain maximal utilization of resources and to minimize the hire cost by sharing resources. The algorithm of task scheduling for HPCaaS cloud (Fig.3) is based on list scheduling [9], where priorities of tasks are based on critical paths, i.e. the highest priority has the task starting the longest path to the sink node, as far as the execution time is considered. Initially, the first application is scheduled using list scheduling. Next, the following task graphs are scheduled. To achieve high utilization of resources and to minimize the number of required resources, the algorithm, in the first order, tries to utilize idle resources used by the previous applications.

```
ScheduleForHPCaaS{
  Makespan=null;
  loop {
    App=getNextApp();
    Time=GetTime();
    List=CreatePriorityList(App);
    while(List!=null){
      task=GetNextTask(List);
      starttime=GetStart(task,Time);
      loop {
        proc=FindIdlePE(starttime);
        if (proc!=null) {
          M’=Add(T,t,P,M);
          if(Length(M’)<=Length(M)+MaxDel(M)){
            Makespan=M’;
            break;
          }
        }
        p=AvailPE();
        if (p!=null) {
          M=Add(T,t,P,M);
          break;
        }
      starttime=NextIdle(starttime);
      }
    }
  }
}
```

Fig. 3. Task scheduling algorithm

Details of the algorithm are as follows. Function *getNextApp()* waits for the next application submitted for execution. *GetTime()* returns current time, it will be stored as a start time of the application. Next, the static priority list of all tasks is created. The subsequent tasks are scheduled in the *while* loop. *GetNextTask()* returns the first task and removes it from the list. *GetStart()* determines the earliest start time for this task. Then, if any idle processor exists (i.e. processor waiting for the next task), the algorithm tries to assign the task to it. This may cause that the next task will be delayed, thus if this delay is shorter than the defined limit (*MaxDel()*), tasks are rescheduled, otherwise task will not be assigned to this processor. In this case if there is any processor with unassigned tasks, then the task will be assigned to this processor. If it is not possible to schedule the task for the current time, the loop will be repeated for the next time frame when any processor will be available (*NextIdle()*) until the task will be scheduled.

Fig. 4a presents the results of the algorithm applied to the task graph from Fig.1. We assumed the system consisting of 2 processors. For the first application the algorithm works as a classical list scheduling. In this case, the priority list is as follows: 3, 2, 4, 5, 11, 6, 7, 10, 8, 9 (source and sink nodes are omitted). Assume that the maximal delay for applications is 4s and assume that another application has been submitted also at time 0 and that it starts with tasks *T0* and *T1* with execution times 5 s and 10 s, respectively. The first available time frame, when any processor is available, is time 6 s (processor *P0* is idle waiting for task 4). The idle time is equal to 1 s, thus task 11 should be delayed. But this does not change the execution time for the whole application, hence *T0* is assigned to *P0* (Fig. 4b). Next, the algorithm tries to schedule *T1* at time 12 s (processor *P0* is waiting for task 10). But in this case, task 10 and consequently tasks 8 will be delayed by 9 s causing that the whole application will finish at time 44 s instead of 36 s. In our case, such a delay is not allowed, thus the algorithm will assign this task to the next time frame (Fig. 4c).

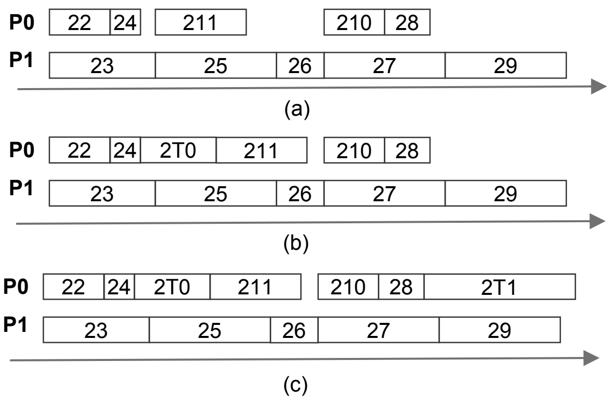

Fig. 4. The task scheduled using list scheduling, a) makespan for the first application, b) makespan after adding task T0, c) makespan after adding task T1

The problem of task scheduling for HPCaaS is a special case of resource constraint project scheduling problems (RCPSP). There are many methods solving different variants of RCPSP [10]. We may define the above problem as the RCPSP with partially available resources. We adopted the list scheduling to this problem, because this method is fast and efficient for scheduling task graphs. Therefore, it may be used during runtime and the time overhead for scheduling will be insignificant.

5. Experimental results

To evaluate our method, we performed some experiments for task graphs from PSP library [11]. We assumed that task graphs were submitted in the following order:

- time 0: c154_3.mm, c159_3.mm,
- time 13: c158_3.mm, c158_4.mm,
- time 43: c159_1.mm, c159_2.mm,

All the tasks graphs consisted of 16 tasks. Table 2 presents the results of scheduling for cluster architecture consisting of 4 processors. First, the classical scheduling for HPCaaS clouds was applied. This method schedules applications on dedicated resources. Since all applications require at least 3 processors, they had to be executed sequentially. The following columns of Tab. 2 contain:

- execution time of the application (schedule length),
- application finishing time,
- utilization of processors,
- latency, i.e. the difference between finishing and submission times.

Next, we applied our method; the last 4 columns of Tab.2 show results. Since our method tries to start the application as soon as possible by utilizing idle processors, makespans for applications are longer than using the classical approach. But the finish time for applications is significantly earlier, thus the latency is shorter i.e. users receive results faster. The average speedup is about 40-50%. Since the cluster requires less time to execute all applications (by 34%), the cost of hiring resources may be significantly reduced.

Tab. 2. Comparison of classical scheduling with the proposed method

Task graph	Classical scheduling				**Our algorithm**			
	Execution time	Finish at	Utilization, %	Latency	Execution time	Finish at	Utilization, %	Latency
c154_3	22	22	56	22	**22**	**22**	**86**	**22**
c159_3	20	42	56	42	**27**	**27**		**27**
c158_3	18	60	74	47	**28**	**41**		**28**
c158_4	19	75	67	62	**32**	**54**		**39**
c159_1	18	97	51	54	**25**	**66**		**23**
c159_2	21	118	52	75	**24**	**78**		**35**

Our method reduces the latency by utilizing idle resources. In our experiments, the utilization of processors was increased from 59% (average utilization for 6 application) to 86%. But in the case of a full load of the cluster, in our method we may obtain significantly higher utilization. As it is shown in Fig.5, the processors are idle mainly during the last time period, when there are no new applications for scheduling.

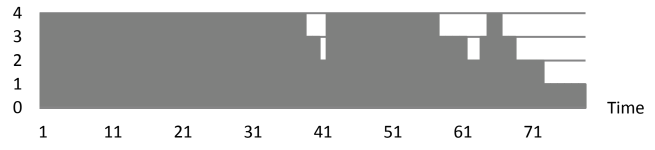

Fig. 5. Utilization of resources for cluster consisting of 4 nodes

6. Conclusions

In this paper, we have presented the method of scheduling applications, specified as task graphs, for HPCaaS clouds. The method is based on the list scheduling and it dynamically schedules applications submitted for execution. It minimizes the cost of hiring resources from the cloud by resource sharing that increases the resource utilization. The experimental results showed that by using our method costs may be reduced by over 30%.

The future work will focus on considering heterogeneous cloud infrastructures as well as other models of distributed applications. We will also modify our approach to take into consideration the communication between tasks. Our method will also be verified by using practical applications and by prototype implementation of HPCaaS cloud.

7. References

[1] AbdelBaky M., Parashar M., Kim H., JordanKirk E. J., Sachdeva V., Sexton J., Jamjoom H., Shae Z.Y., Pencheva G., Tavakoli R., Wheeler M. F.: Enabling High Performance Computing as a Service. IEEE Computer, vol.45, no.10, pp. 72-80, 2012.

[2] Gupta A., Faraboschi P., Gioachin F., Kale L.V., Kaufmann R., Lee B.S., March V., Milojicic D., Suen C.H.: Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud. IEEE Transactions on Cloud Computing, vol. 99, 2014.

[3] Shainer G., Liu T., Layton J., Mora J.: Scheduling strategies for HPC as a service (HPCaaS). Proc. IEEE International Conference on Cluster Computing and Workshops, 2009.

[4] Somasundaram T.S., Govindarajan K.: CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud. Future Generation Computer Systems, vol. 34, pp.47-65, 2014.

[5] Jamalian S., Rajaei H.: Data-Intensive HPC Tasks Scheduling with SDN to Enable HPC-as-a-Service. Proc. IEEE 8th International Conference on Cloud Computing (CLOUD), pp.596-603, 2015.

[6] Huang K.C., Huanga T.C., Tsaia M.J., Chang H.Y., Tung Y.H.: Moldable job scheduling for HPC as a service with application speedup model and execution time information. Journal of Convergence, vol.4, no.4, 2013.

[7] Narayana S. S., Batrashev O., Jakovits P., Vainikko E.: Scalability of parallel scientific applications on the cloud. Scientific Programming, vol.19, no.2-3, pp.91-105, 2011.

[8] Bąk S., Czarnecki R., Deniziak S.: Scheduling of Distributed Applications for High Performance Computing as a Service. Proc International Conference on Computational Methods in Sciences and Engineering, 2016.

[9] Sinnen O.: Task Scheduling for Parallel Systems. John Wiley, 2007.

[10] Hartmann S., Briskorn D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European journal of operational research: EJOR. – Amsterdam, Elsevier, Vol. 207, 1 (16.11.), pp. 1-15, 2010.

[11] Kolisch R., Sprecher A.: PSPLIB - A project scheduling library. European Journal of Operational Research, Vol. 96, pp. 205-216, 1996.

Received: 18.03.2016 *Paper reviewed* *Accepted: 02.05.2016*

Stanisław DENIZIAK, DSc, eng.

He graduated on Faculty of Electronics of the Warsaw University of Technology, defended his doctoral thesis in 1994, and habilitation in 2006. He is the head of the Division of Computer Science of the Kielce University of Technology. His research interests include design of embedded systems, Internet of things, distributed computing, logic synthesis for FPGAs. He is a member of IEEE and IEEE Computer Society.

e-mail: S.Deniziak@computer.org

Paweł PADUCH, PhD, eng.

He graduated on Faculty of Electrical Automation and Computer Science of the Kielce University of Technology in 2002, defended his doctoral thesis on Military University of Technology in Warsaw in 2012. He is the assistant professor on the Division of Computer Science of the Kielce University of Technology. He is interested in ants algorithms, nature inspired computation and parallel and distributed computing.

e-mail: paduch@tu.kielce.pl