Conference Paper

Scheduling in Cloud Computing Environment

Amine Mahjoub and Youssef Harrath

College of Information Technology, University of Bahrain

Abstract

In this work, the authors focus on the problem of scheduling in cloud computing environment. Such environment is characterized by a very strong volatility, in the sense that the resources can integrate and leave the system in a frequent and completely random way. The classical scheduling problems consider that the resources are always available. This hypothesis is not realistic especially in cloud computing environment. In this article, the authors present an efficient algorithm to solve the problem of scheduling in a simplified model of a cloud environment where the resources are subject to deterministic unavailability periods.

Keywords: Parallel machine scheduling, Resource unavailability, List scheduling; LPT.

Corresponding Author:
Amine Mahjoub
amahjoub@uob.edu.bh

Received: 18 September 2018
Accepted: 10 October 2018
Published: 15 October 2018

Publishing services provided by Knowledge E

1. Introduction and Motivation

Cloud computing is a recent and important topic in the Computer Science field. Considerable efforts continue to be produced to develop techniques and efficient solutions to many emerging problems in this topic. Cloud computing nowadays is an IT-related service provider that allows to get many possible IT services at one place. The most common services provided by a cloud are:

1. SaaS or Software as a service: Allow the users to temporarily get profit of available software without the need of personal license.
2. Utility Computing: This is probably the most important feature of a cloud computing where the users can access virtual storages, as well as purchase computing capacities to remotely perform super calculation tasks.
3. Platforms: This is a more advanced feature of cloud computing service where complete platforms providing whole development environment are made available for users.

In this paper, we are interested in utility computing services of a cloud, where computing resources are available to execute users' tasks. The main goal is to execute the customers' applications as early as possible with the lowest cost. The particularity of the cloud computing environment is that the resources are not continuously available. Thus, the execution of an application in the cloud can be seen as a scheduling problem of independents tasks on parallel machines with unavailability periods.

The proposed research is focused on developing an efficient solution to the scheduling problem on parallel and identical machines subject to unavailability constraints. The objective is to optimize the completion execution time of the application. Classically, an application is composed by a set of independent tasks, each has a processing time. The scheduling problem consists of determining on which processor each task has to be executed and at what time it starts so that the last executed task terminates as early as possible. In this paper we propose a new efficient linear program, and will prove that LPT-based list algorithm based provides performant solutions.

The rest of the paper is organized as follows. Section 2 will be dedicated to present an overview of the literature. In section 3 an efficient linear program will be proposed. An experimental study will be shown in section 4. Section 5 concludes the study proposes some perspectives for future works.

2. Literature Review

The most parallel machine scheduling problem was proved to be NP-complete [2]. However, several heuristics based on different techniques, have been proposed in the literature. Among these, list schedules are paramount [1]. List schedules can be adapted to non-continuously available machines (due to unavailability periods or breakdowns).

A list algorithm, first establishes a priority list. At each time *t*, if some machines M_i are available, the first non-scheduled task *j* of the list is placed on the machine if it can be executed before the machine becomes unavailable. If *j* cannot be entirely executed, the next task of the list is considered, and so on. Hence, contrary to the case without unavailability, a task with smaller priority can be executed before a task of larger priority. When the list of task is ordered in decreasing order of processing times, the obtained algorithm is called LPT algorithm.

For the remaining of this paper, *m* and *n* represent respectively the number of machines and the number of tasks. The scheduling problem of m machines with unavailability constraints is widely studied under different constraints such as resumable/non-resumable jobs, offline/online scheduling and various objective functions including minimizing the maximum completion time (makespan), total completion times, etc. ([3], [4], [5], and [6]).

Lee [7] showed that, if the unavailabilities are at the beginning of the execution period (that is, machines are not all available at time 0), *LPT* has a relative bound of $\frac{3}{2} - \frac{1}{2m}$. A survey on scheduling problems with machine unavailabilities may be found in [9], [10] and [8].

In this paper, we consider *n* independent tasks to be scheduled on *m* identical machines. Each task *j* has a processing time p_j. The machines are subject to many unavailability periods each of them has a starting and ending times. We denote by C_{LS} the makespan of the schedule obtained by any list algorithm *LS*. C^* represents the optimal makespan. We denote by $P = Max_{1\leq j\leq n}(p_j)$ the maximum processing time of tasks.

3. Linear Program

We consider the case of many unavailability periods per machine. This means that every machine is subject to at least one unavailability period and there exists at least one machine, which is subject to at least two unavailability periods. For a machine M_i, we denote by s_{ik} and e_{ik}, respectively the starting and ending times of the k^{th} unavailability period, $1 \leq k \leq u_i$ $1 \ \leq k \leq u_i$, where u_i is the number of unavailabilities in machine $M_{i.}$

The problem under consideration is equivalent to a multiple knapsack problem with different capacities. In fact, each machine has a sequence of availability periods regularly interrupted by random and predefined events. Each availability period can be seen as a knapsack with a capacity equals to the difference between its ending and starting times (e_{ik}) and (s_{ik}). The tasks represent the items, where the item size is the duration of a task. Since the tasks have no priorities, the items will be assigned the same value.

The objective is to assign the maximum of tasks in the most left knapsacks according to the time line. Classically, the linear programs proposed to formulate a scheduling problem, care about assigning tasks to machines as well as find the sequencing of tasks in every machine. To do so, two family of decision variables have to be used. In the proposed linear program, the machines are not anymore directly considered. Thus, the order of tasks in every machine is not any more important. As a fact, the complexity of the developed linear program is extremely reduced. We denote by m_k the total number of knapsacks (availability periods) and K_{ps} an array of size m_k containing the staring times of the knapsacks. Similarly, we denote by K_{pc} an array of size m_k that contains the capacities of the knapsacks. K_{ps} and K_{pc} are parallel arrays. Consider the following decision variables:

$$x_{kj} = \begin{cases} 1 \text{ if the task } j \text{ is assigned to the knapsack } k \\ 0 \text{ otherwise} \end{cases}$$

$$y_k = \begin{cases} 1 \text{ if at least on task is assigned to the knapsack } k \\ 0 \text{ otherwise} \end{cases}$$

Where $1 \leq j \leq n$ and $1 \leq k \leq m_k$

The objective function can be summarized as:

$$Min \;\; z = \underset{1 \leq k \leq m_k}{Max} \left(y_k K_{ps}[k] + \sum_{j=1}^{n} x_{kj} p_j \right) \tag{1}$$

Subject to the constraints:

$$y_k \leq 1 \quad \forall \; 1 \leq k \leq m_k \tag{2}$$

$$y_k \leq \sum_{j=1}^{n} x_{kj} \quad \forall \; 1 \leq k \leq m_k \tag{3}$$

$$y_k \geq x_{kj} \quad \forall \; 1 \leq k \leq m_k \text{ and } 1 \leq j \leq n_k \tag{4}$$

$$\sum_{j=1}^{n} x_{kj} p_j \leq K_{pc}[k] \quad \forall \; 1 \leq k \leq m_k \tag{5}$$

$$\sum_{k=1}^{m_k} x_{kj} \leq 1 \quad \forall \; 1 \leq j \leq n \tag{6}$$

The constraints (2), (3) and (4) guarantee that if there is at least one task assigned to a period *k*, then the corresponding decision variable $y_k = 1$ Otherwise $y_k = 0$.

4. Experimental study

The objective of this section is to test the performance of the *LPT* algorithm for the problem as well as the limitations of the proposed linear program. We have implemented the following *LPT* rule: Sort the tasks in a decreasing order of their processing times. The first task of the list is assigned to the first available machine. If that task cannot be executed because of an unavailability period, the next task of the list is examined. If two machines are simultaneously available, one of them is chosen randomly. In the other hand, we implemented the linear program using ILOG CPLEX 12.7. To test both the *LPT* algorithm and the linear program, a wide range of instances of different sizes were uniformly generated as follows.

1. The processing times of the tasks are in the interval $[p, P]$. Experimentally, p and P were respectively set to the values 10 and 20.
2. The durations of the availability periods are in $[P, 2P]$.
3. The durations of the unavailability periods are in $[p, 2P]$.
4. The number of machines is 4 for the small-sized instances where the number of tasks varies between 20 and 70. However, for the big-sized instances, where the number of tasks varies between 500 and 1000, the number of machines is fixed to 10. Based on a deep conducted experimental study, where we have varied the number of machines m, we remarked that m has no significant effect on the *LPT* performance. For this reason, we fixed the number of machines.
5. For both, small and big-sized instances, we have varied the number of unavailabilities between 3 and 5.

To compare the *LPT* solutions and the optimal ones, we define an error ratio $e_r = \frac{C_{LPT}}{C*}$. Table 1 and Table 2 show the different results obtained and the error ratio between *LPT* and the optimal solution for big and small size instances.

For both configurations, *LPT* performs well and has almost similar behavior toward the optimal solution provided by the linear program. In fact, the completion time C_{LPT} is very close to the optimal solution. In addition, *LPT* seems to have slightly better performance for the big sized-instances. Furthermore, the error ratio slightly increases when the unavailabilities increase but still small.

The obtained experimental results prove the fact that although the problem is strongly NP-hard, it can be solved efficiently by a simple and polynomial *LPT*-based list algorithm. The experimental study showed also, that the developed linear program is very efficient. Indeed, for large instances with more than 20 machines and 2000 tasks, CPLEX returns the solutions in few seconds.

Table 1: Error ratios for big size instances.

	3 Unavailabilities			4 Unavailabilities			5 Unavailabilities		
Number of tasks	LPT	Optimal	Ratio	LPT	Optimal	Ratio	LPT	Optimal	Ratio
500	780	770	1.3	836	821	1.8	850	835	1.8
520	803	792	1.4	866	852	1.6	869	856	1.5
540	835	823	1.5	901	888	1.5	899	888	1.2
560	862	854	0.9	927	916	1.2	935	921	1.5
580	898	889	1.0	957	948	0.9	968	954	1.5
600	926	916	1.1	990	979	1.1	994	981	1.3
620	957	944	1.4	1018	1004	1.4	1024	1010	1.4
640	992	980	1.2	1050	1033	1.6	1048	1034	1.4
660	1012	1000	1.2	1075	1060	1.4	1084	1072	1.1
680	1036	1026	1.0	1100	1085	1.4	1119	1100	1.7
700	1070	1058	1.1	1127	1111	1.4	1143	1124	1.7
720	1096	1084	1.1	1151	1137	1.2	1170	1155	1.3
740	1130	1121	0.8	1184	1161	2.0	1201	1182	1.6
760	1147	1140	0.6	1210	1195	1.3	1218	1207	0.9
780	1189	1178	0.9	1235	1220	1.2	1261	1246	1.2
800	1222	1211	0.9	1259	1246	1.0	1292	1276	1.3
820	1247	1235	1.0	1296	1280	1.3	1326	1312	1.1
840	1281	1269	0.9	1318	1308	0.8	1356	1341	1.1
860	1297	1286	0.9	1348	1336	0.9	1382	1365	1.2
880	1342	1330	0.9	1368	1357	0.8	1404	1389	1.1
900	1347	1337	0.7	1402	1391	0.8	1433	1416	1.2
920	1397	1385	0.9	1441	1431	0.7	1466	1449	1.2
940	1414	1403	0.8	1473	1464	0.6	1492	1479	0.9
960	1453	1438	1.0	1499	1490	0.6	1530	1514	1.1
980	1486	1475	0.7	1531	1519	0.8	1560	1544	1.0
1000	1502	1494	0.5	1560	1550	0.6	1585	1572	0.8

5. Conclusion

In this paper, we have considered the scheduling problem of independents tasks on identical parallel machines. The machines are subject to many unavailability periods. The objective was to minimize the ending time of the last task or the makespan. A new linear program was designed and tested to solve variety of instances of different sizes. This program served for proving that and LPT-based algorithm can solve the problem in an efficient way. This work is a first part of a global project where the goal consists of producing efficient algorithms to schedule and manage online parallel applications in cloud computing environment.

Table 2: Error ratios for small size instances.

	3 Unavailabilities			4 Unavailabilities			5 Unavailabilities		
Number of tasks	**LPT**	**Optimal**	**Ratio**	**LPT**	**Optimal**	**Ratio**	**LPT**	**Optimal**	**Ratio**
19	132	114	15.8	130	130	0.0	127	116	9.5
21	142	125	13.6	146	138	5.8	129	121	6.6
23	165	124	33.1	157	155	1.3	170	147	15.6
25	173	162	6.8	163	154	5.8	174	157	10.8
27	209	193	8.3	191	180	6.1	201	170	18.2
29	206	196	5.1	203	183	10.9	191	181	5.5
31	201	195	3.1	224	195	14.9	230	191	20.4
33	215	198	8.6	233	216	7.9	232	212	9.4
35	222	209	6.2	242	223	8.5	250	232	7.8
37	229	220	4.1	235	225	4.4	259	242	7.0
39	226	221	2.3	247	230	7.4	282	263	7.2
41	236	225	4.9	253	235	7.7	296	275	7.6
43	232	226	2.7	254	244	4.1	308	286	7.7
45	249	238	4.6	272	253	7.5	315	298	5.7
47	263	250	5.2	277	262	5.7	320	302	6.0
49	273	263	3.8	291	272	7.0	331	313	5.8
51	270	256	5.5	284	268	6.0	337	324	4.0
53	266	256	3.9	299	282	6.0	349	329	6.1
55	275	264	4.2	303	294	3.1	350	329	6.4
57	285	272	4.8	321	306	4.9	356	337	5.6
59	284	276	2.9	329	314	4.8	362	346	4.6
61	294	282	4.3	329	318	3.5	364	348	4.6
63	299	292	2.4	337	316	6.6	360	350	2.9
65	326	314	3.8	332	317	4.7	382	368	3.8
67	320	315	1.6	345	333	3.6	390	375	4.0
69	328	319	2.8	347	336	3.3	403	380	6.1

References

[1] R.L. Graham, Bounds on multiprocessing timing anomalies, *SIAM Journal of Applied Mathematics*, Vol. 17, No. 2, pp. 416-429, 1969.

[2] M.R. Carey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman publisher, San Francisco, CA, 1979.

[3] A. Berrichi, F. Yalaoui, L. Amodeo, M. Mezghiche, Bi-Objective Ant Colony Optimization approach to optimize production and maintenance scheduling, *Computers and Operations Research*, Vol. 37, pp. 1584-1596, 2010.

[4] Bin Fu, Yumei Huo, Hairong Zhao, Approximation schemes for parallel machine scheduling with availability constraints, *Discrete Applied Mathematics*, Vol. 159, pp. 1555-1565, 2011.

[5] Ming Liu, Feifeng Zheng, Chengbin Chu, Yinfeng Xu, Optimal algorithms for online scheduling on parallel machines to minimize the makespan with a periodic availability constraint, *Theoretical Computer Science*, Vol. 42, pp. 5225-5231, 2011.

[6] Zhiyi, Parallel machines scheduling with machine maintenance for minsum criteria, *European Journal of Operational Research*, Vol. 212, pp. 287-292, 2011.

[7] Chung-Yee Lee, Parallel machines scheduling with nonsimultaneous machine available time, *Discrete Applied Mathematics*, V. 30, pp. 53-61, 1991.

[8] J. Kaabi, Y. Harrath, A Survey of Parallel Machine Scheduling under Availability Constraints, *International Journal of Computer and Information Technology*, Vol. 3, No 2, pp. 238-245, 2014

[9] Y. Ma, C. Chu, C. Zuo, A survey of scheduling with deterministic machine availability constraints, *Computers and Industrial Engineering*, Vol. 58, No. 2, pp. 199-211, 2010.

[10] E. Sanlaville, G. Schmidt, Machine scheduling with availability constraints, *Acta Informatica*, Vol. 35, pp. 795-811, 1998.