Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments

Hesam Izakian[1], Ajith Abraham[2], *Senior Member, IEEE*, Václav Snášel[3]
[1] Islamic Azad University, Ramsar Branch, Ramsar, Iran
[2] Norwegian Center of Excellence, Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology, Trondheim, Norway
[3]Faculty of Electrical Engineering and Computer Science VSB-Technical University of Ostrava, Czech Republic
Hesam.izakian@gmail.com, Ajith.abraham@ieee.org, vaclav.snasel@vsb.cz

Abstract

Scheduling is one of the core steps to efficiently exploit the capabilities of heterogeneous distributed computing systems and is an NP-complete problem. Therefore using meta-heuristic algorithms is a suitable approach in order to cope with its difficulty. In meta-heuristic algorithms, generating individuals in the initial step has an important effect on the convergence behavior of the algorithm and final solutions. Using some heuristics for generating one or more near-optimal individuals in the initial step can improve the final solutions obtained by meta-heuristic algorithms. Different criteria can be used for evaluating the efficiency of scheduling algorithms, the most important of which are makespan and flowtime. In this paper we propose an efficient heuristic method and then we will compare with five popular heuristics for minimizing makespan and flowtime in heterogeneous distributed computing systems.

1. Introduction

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links, to perform different computationally intensive applications that have diverse computational requirements [1, 2]. To exploit the different capabilities of a suite of heterogeneous resources, typically a resource management system (RMS) allocates the resources to the tasks and the tasks are ordered for execution on the resources. At a time interval in HC environment a number of tasks are received by RMS from different users. Different tasks have different requirements and different resources have different capabilities. Optimally scheduling is mapping a set of tasks to a set of resources to efficiently exploit the capabilities of such systems and is one of the key problems in HC environments. As mentioned in [9] optimal mapping tasks to machines in an HC suite is an NP-complete problem and therefore the use of meta-heuristics is one of the suitable approaches. The most popular of meta-heuristic algorithms are genetic algorithm (GA), tabu search (TS), simulated annealing (SA), ant colony optimization (ACO) and particle swarm optimization (PSO).

Ritchie and Levine [4] used a hybrid ant colony optimization, Yarkhan and Dongarra [5] used simulated annealing approach and Page and Naughton [3], used genetic algorithm for task scheduling in HC systems.

The algorithmic flow in meta-heuristic algorithms starts with randomly generating population of individuals that are potential solutions. Then in a fixed number of iterations the algorithm tries to obtain optimal or near-optimal solutions using predefined operators (such as crossover and mutation in GA etc) and a fitness function that evaluates the optimality of solutions. Generating potential solutions at the beginning of the algorithm has an important effect in obtaining final solutions and if in this step of the algorithm bad solutions are generated randomly, then the algorithm provides bad solutions or local optimal solutions. To overcome the posed problem, we usually generate one or more individuals using well-known heuristics and others randomly in the initial step of the algorithm. These heuristics generate near-optimal

978-0-7695-3605-7/09 $25.00 © 2009 IEEE
DOI 10.1109/CSO.2009.487

solutions and the meta-heuristic algorithm combines random solutions with them for obtaining better solutions. Using this method we can obtain better solutions using meta-heuristic algorithms.

Existing scheduling heuristics can be divided into two classes [6]: on-line mode (immediate mode) and batch-mode heuristics. In the on-line mode, a task is mapped onto a host as soon as it arrives at the scheduler. In the batch mode, tasks are not mapped onto hosts immediately and they are collected into a set of tasks that is examined for mapping at prescheduled times called mapping events. The online mode heuristic is suitable for the low arrival rate, while batch-mode heuristics can achieve higher performance when the arrival rate of tasks is high because there will be a sufficient number of tasks to keep hosts busy between the mapping events, and scheduling is according to the resource requirement information of all tasks in the set [6]. In this paper, we considered batch-mode heuristics.

Different criteria can be used for evaluating the efficiency of scheduling algorithms, the most important of which are makespan and flowtime. Makespan is the time when an HC system finishes the latest job and flowtime is the sum of finalization times of all the jobs. An optimal schedule will be the one that optimizes the flowtime and makespan.

In this paper, we proposed an efficient heuristic called min-max. Also we investigate the efficacy of min-max and 5 popular heuristics for minimizing makespan and flowtime. These heuristics are min-min, max-min, LJFR-SJFR, sufferage, and WorkQueue. These heuristics are popular, effective and are used in many studies. So far, some of works have been done for investigating number of these heuristics for minimizing makespan, yet no attempt has been made to minimize flowtime or both flowtime and makespan. Also the efficiency of these heuristics is investigated on simple benchmarks and the various characteristics of machines and tasks in HC environments are not considered. In this paper, we investigate the efficiency of these heuristics on HC environments with various characteristics of both machines and tasks.

The remainder of this paper is organized in the following manner: Section 2 formulates the problem, in Section 3 we provide the definitions of heuristics, and Section 4 reports the experimental results. Finally Section 5 concludes this work.

2. Problem formulation

An HC environment is composed of computing resources where these resources can be a single PC, a cluster of workstations or a supercomputer. Let $T = \{T_1, T_2, ..., T_n\}$ denote the set of tasks that in a specific time interval is submitted to RMS. Assume the tasks are independent of each other (with no inter-task data dependencies) and preemption is not allowed (they cannot change the resource they have been assigned to). Also assume at the time of receiving these tasks by RMS, m machines $M = \{M_1, M_2, ..., M_m\}$ are within the HC environment. In this paper scheduling is done at machine level and it is assumed that each machine uses First-Come, First-Served (FCFS) method for performing the received tasks. We assume that each machine in HC environment can estimate how much time is required to perform each task. In [2] Expected Time to Compute (ECT) matrix is used to estimate the required time for executing a task in a machine. An ETC matrix is an $n \times m$ matrix in which n is the number of tasks and m is the number of machines. One row of the ETC matrix contains the estimated execution time for a given task on each machine. Similarly one column of the ETC matrix consists of the estimated execution time of a given machine for each task. Thus, for an arbitrary task T_j and an arbitrary machine M_i, $ETC(T_j, M_i)$ is the estimated execution time of T_j on M_i. In ETC model we take the usual assumption that we know the computing capacity of each resource, an estimation or prediction of the computational needs of each job, and the load of prior work of each resource.

Assume that $C_{i,j}$ $(i \in \{1,2,...,m\}, j \in \{1,2,...,n\})$ is the completion time for performing *j*th task in *i*th machine and W_i $(i \in \{1,2,...,m\})$ is the previous workload of M_i, then Eq. (1) shows the time required for M_i to complete the tasks included in it. According to the aforementioned definition, makespan and flowtime can be estimated using Eq. (2) and Eq. (3) respectively.

$$\sum C_i + W_i \tag{1}$$

$$makespan = \max\{\sum C_i + W_i\}, \quad i \in \{1,2,...,m\} \tag{2}$$

$$flowtime = \sum_{i=1}^{m} C_i \tag{3}$$

As mentioned in the previous section, the goal of the scheduler in this paper is to minimize makespan and flowtime.

3. Heuristic descriptions

This section provides the description of 5 popular heuristics for mapping tasks to available machines in HC environments. Then we propose an efficient heuristic called min-max.

3.1. Min-min heuristic

Min-min heuristic uses minimum completion time (MCT) as a metric, meaning that the task which can be completed the earliest is given priority. This heuristic begins with the set U of all unmapped tasks. Then the set of minimum completion times, $M = \{\min(completion\_time(T_i, M_j)) \quad for\ (1 \le i \le n,\ 1 \le j \le m)\}$, is found. M consists of one entry for each unmapped task. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine and the workload of the selected machine will be updated. And finally the newly mapped task is removed from U and the process repeats until all tasks are mapped (i.e. U is empty) [2, 7].

3.2. Max-min heuristic

The Max-min heuristic is very similar to min-min and its metric is MCT too. It begins with the set U of all unmapped tasks. Then, the set of minimum completion times, $M = \{\min(completion\_time(T_i, M_j))$, $for\ (1 \le i \le n,\ 1 \le j \le m)\}$, is found. Next, the task with the overall maximum completion time from M is selected and assigned to the corresponding machine and the workload of the selected machine will be updated. And finally the newly mapped task is removed from U and the process repeats until all tasks are mapped [2, 7].

3.3. LJFR-SJFR Heuristic

Longest Job to Fastest Resource- Shortest Job to Fastest Resource (LJFR-SJFR) [8] heuristic begins with the set U of all unmapped tasks. Then, the set of minimum completion times, $M = \{\min(completion\_time(T_i, M_j)) \quad for\ (1 \le i \le n,\ 1 \le j \le m)\}$, is found the same as min-min. Next, the task with the overall minimum completion time from M is considered as the shortest job in the fastest resource (SJFR). Also the task with the overall maximum completion time from M is considered as the longest job in the fastest resource (LJFR). At the beginning, this method assigns the m longest jobs to the m available fastest resources (LJFR) and then assigns the shortest task to the fastest resource and the longest task to the fastest resource alternatively. After each allocation, the workload of each machine will be updated.

3.4. Sufferage Heuristic

In this heuristic for each task, the minimum and second minimum completion time are found in the first step. The difference between these two values is defined as the sufferage value. In the second step, the task with the maximum sufferage value is assigned to the corresponding machine with minimum completion time. The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it [6].

3.5. WorkQueue Heuristic

This heuristic is a straightforward and adaptive scheduling algorithm for scheduling sets of independent tasks. In this method the heuristic selects a task randomly and assigns it to the machine as soon as it becomes available (in other word the machine with minimum workload).

3.6. Proposed Heuristic

This heuristic (called min-max) is composed of two steps for mapping each task and uses the minimum completion time in the first step and the minimum execution time in the second as metric. In the first step, this heuristic begins with the set U of all unmapped tasks. Then the set of minimum completion times, $M = \{\min(completion\_time(T_i, M_j))$ $for\ (1 \le i \le n,\ 1 \le j \le m)\}$, is found the same as min-min heuristic. In the second step, the task whose minimum execution time (time for executing task on the fastest machine) divide by its execution time on the selected machine (in the first step), has the maximum value will be selected for mapping. The intuition behind this heuristic is that we select pair machines and tasks from the first step that the

machine can executes its corresponding task effectively with a lower execution time in comparison with other machines.

4. Comparison and Experimental results

We compared the performance of the above heuristics for minimizing makespan and flowtime. We used the benchmark proposed in [2]. The simulation model in [2] is based on expected time to compute (ETC) matrix for 512 jobs and 16 machines. The instances of the benchmark are classified into 12 different types of ETC matrices according to the three following metrics: job heterogeneity, machine heterogeneity, and consistency. In ETC matrix, the amount of variance among the execution times of tasks for a given machine is defined as task heterogeneity. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Also an ETC matrix is said to be consistent whenever a machine M_j executes any task T_i faster than machine M_k; in this case, machine M_j executes all tasks faster than machine M_k. In contrast, inconsistent matrices characterize the situation where machine M_j may be faster than machine M_k for some tasks and slower for others. Partially-consistent matrices are inconsistent matrices that include a consistent sub-matrix of a predefined size [2].

Instances consist of 512 jobs and 16 machines and are labeled as u-yy-zz-x as follow:

- u means uniform distribution used in generating the matrices.
- yy indicates the heterogeneity of the jobs; hi means high and lo means low.
- zz represents the heterogeneity of the nodes; hi means high and lo means low.
- x shows the type of inconsistency; c means consistent, i means inconsistent, and p means partially-consistent.

The obtained makespan and flowtime using mentioned heuristics are compared in tables 1 and 2 respectively. The results are obtained as an average of five simulations. In these tables, the first column indicates the instance name, and the second, third, fourth, fifth and sixth columns indicate the makespan and flowtime of workQueue, max-min, LJFR-SJFR, Sufferage, min-min and min-max heuristics.

Figures 1 and 2 show the comparison of statistical results using different heuristics for mean makespan and flowtime for the 12 considered cases. As it is evident from the figures, min-max, the proposed heuristic, can minimize the makespan better than others in most cases. Also min-min heuristic can minimize flowtime better than others.

5. Conclusions

Scheduling in HC environments is an NP-complete problem. Therefore, using meta-heuristic algorithms is a suitable approach in order to cope with its difficulty in practice. In meta-heuristic algorithms, the use of one or more heuristics for generating individuals is an appropriate method that can improve the final solutions. In this paper we compare 6 heuristics for scheduling in HC environments. The goal of the scheduler in this paper is minimizing makespan and flowtime. The experimental results show that min-min heuristic can obtain the best results for minimizing flowtime and the proposed heuristic can obtain the best results for minimizing makespan too. These results indicate that using min-max heuristic for generating initial individuals in meta-heuristic algorithms is a suitable selection.

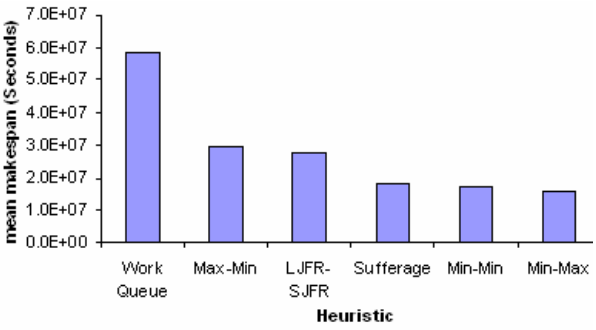

Figure 1. Comparison results between heuristics on makespan

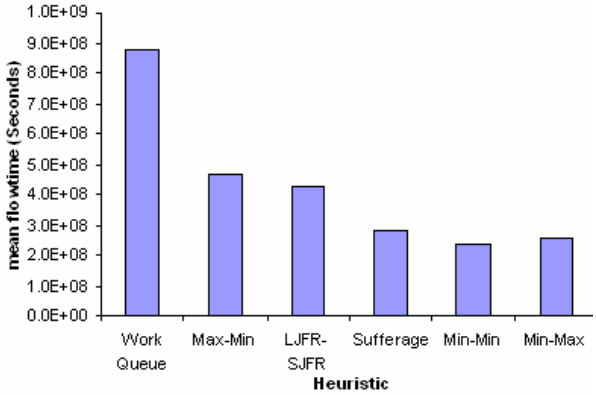

Figure 2. Comparison results between heuristics on flowtime

Table 1. Comparison of statistical results on makespan (Seconds)

Instance	WorkQueue	Max-Min	LJFR-SJFR	Sufferage	Min-Min	Min-Max
u-lo-lo-c	7332	6753	6563	5461	5468	**5310**
u-lo-lo-p	8258	5947	5179	3433	3599	**3327**
u-lo-lo-i	9099	4998	4251	2577	2734	**2523**
u-lo-hi-c	473353	400222	391715	333413	279651	**273467**
u-lo-hi-p	647404	314048	279713	163846	157307	**146953**
u-lo-hi-i	836701	232419	209076	121738	113944	**102543**
u-hi-lo-c	203180	203684	202010	170663	164490	**164134**
u-hi-lo-p	251980	169782	155969	105661	106322	**103321**
u-hi-lo-i	283553	153992	138256	**77753**	82936	77873
u-hi-hi-c	13717654	11637786	11305465	9228550	8145395	**7878374**
u-hi-hi-p	18977807	9097358	8027802	4922677	4701249	**4368071**
u-hi-hi-i	23286178	7016532	6623221	3366693	3573987	**2989993**

Table 2. Comparison of statistical results on flowtime (Seconds)

Instance	WorkQueue	Max-Min	LJFR-SJFR	Sufferage	Min-Min	Min-Max
u-lo-lo-c	108843	108014	102810	86643	**80354**	84717
u-lo-lo-p	127639	95091	81861	54075	**51399**	52935
u-lo-lo-i	140764	79882	66812	40235	**39605**	39679
u-lo-hi-c	7235486	6400684	6078313	5271246	**3918515**	4357089
u-lo-hi-p	10028494	5017831	4383010	2568300	**2118116**	2323396
u-lo-hi-i	12422991	3710963	3303836	1641220	**1577886**	1589574
u-hi-lo-c	3043653	3257403	3153607	2693264	**2480404**	2613333
u-hi-lo-p	3776731	2714227	2461337	1657537	**1565877**	1640408
u-hi-lo-i	4382650	2462485	2181042	1230495	1214038	**1205625**
u-hi-hi-c	203118678	185988129	173379857	145482572	**115162284**	125659590
u-hi-hi-p	282014637	145337260	126917002	76238739	**63516912**	69472441
u-hi-hi-i	352446704	112145666	104660439	47237165	**45696141**	46118709

References

[1] S. Ali, T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Heterogeneous computing", Encyclopedia of Distributed Computing, Kluwer Academic, 2001.

[2] H.J. Braun et al, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems" Journal of Parallel and Distributed Computing, 61(6), 2001.

[3] J. Page and J. Naughton, "Framework for task scheduling in heterogeneous distributed computing using genetic algorithms", Artificial Intelligence Review, 2005 pp. 415–429.

[4] G. Ritchie and J. Levine, "A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments", In: 23rd Workshop of the UK Planning and Scheduling Special Interest Group, 2004.

[5] A. Yarkhan and J. Dongarra, "Experiments with scheduling using simulated annealing in a grid environment", In: 3rd International Workshop on Grid Computing (GRID2002), 2002, pp. 232–242.

[6] M. Macheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems", J. Parallel Distribut. Comput. 59 (2) (1999) 107–131.

[7] R. F. Freund et al, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet", In: 7th IEEE Heterogeneous Computing Workshop (HCW 98), 1998, pp. 184-199.

[8] A. Abraham, R. Buyya, and B. Nath, "Nature's heuristics for scheduling jobs on computational grids", In: The 8th IEEE International Conference on Advanced Computing and Communications, India, 2000.

[9] D. Fernandez-Baca, "Allocating modules to processors in a distributed system", IEEE Trans. Software Engrg. 15, 11 (Nov. 1989), pp. 1427-1436.