Future Generation Computer Systems 60 (2016) 48–66

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Scheduling independent tasks on heterogeneous processors using heuristics and Column Pricing

Christos Gogos [a,*], Christos Valouxis [b], Panayiotis Alefragis [c], George Goulas [c], Nikolaos Voros [c], Efthymios Housos [d]

[a] Technological Educational Institute of Epirus, Department of Computer and Informatics Engineering, Arta, Greece
[b] Ministry of Education, Achaia Branch Office, Patras, Greece
[c] Technological Educational Institute of Western Greece, Department of Computer and Informatics Engineering, Patras, Greece
[d] University of Patras, Department of Electrical and Computer Engineering, Patras, Greece

ARTICLE INFO

Article history:
Received 30 May 2015
Received in revised form
28 December 2015
Accepted 24 January 2016
Available online 3 February 2016

Keywords:
Heterogeneous processors
Independent tasks
Task scheduling
Heuristics
Mathematical programming
Column pricing

ABSTRACT

Efficiently scheduling a set of independent tasks on a virtual supercomputer formed by many heterogeneous components has great practical importance, since such systems are commonly used nowadays. Scheduling efficiency can be seen as the problem of minimizing the overall execution time (makespan) of the set of tasks under question. This problem is known to be NP-hard and is currently addressed using heuristics, evolutionary algorithms and other optimization methods. In this paper, firstly, two novel fast executing heuristics, called LSufferage and TPB, are introduced. L(ist)Sufferage is based on the known heuristic Sufferage and can achieve in general better results than it for most of the cases. T(enacious)PB is also based on another heuristic (Penalty Based) and incorporates new ideas that significantly improve the quality of the resulted schedule. Secondly, a mathematical model of the problem is presented alongside with an associated approach based on the Linear Programming method of Column Pricing. This approach, which is called Column Pricing with Restarts (CPR), can be categorized as a hybrid mathematical programming and heuristic approach and is capable of solving in reasonable time problem instances of practically any size. Experiments show that CPR achieves superior results improving over published results on problem instances of various sizes. Moreover, hardware requirements of CPR are minimal.

1. Introduction

During the last decades technology progress transformed computing power from a property of the few to an asset that can be used by virtually anybody. Homogeneous systems, which refers to multiprocessor systems consisting of identical processing units, were the first wave of systems that was build in order to serve as a processing infrastructure capable of parallel execution for a set of tasks. These tasks typically have execution dependencies among them captured in task graphs. The second wave was heterogeneous systems, consisting of several interconnected computers without common hardware characteristics running under a resource coordination software. Grid computing [1] has been established as the general term when referring to such systems. Under the promise that in a cost effective manner they can execute multiple processing tasks of varying complexity, they have become extremely popular. The problem of task scheduling is critical in homogeneous systems and grid computing [2], since substandard task-processor allocations result to performance losses. For the case of homogeneous systems, heuristics and techniques mainly originated from Operations Research were proposed in order to address the problem of scheduling [3–5]. Likewise, several techniques were proposed for the scheduling problem on grid computing systems [6–8].

In this paper the scheduling problem addressed is the Heterogeneous Computing Scheduling Problem (HCSP), which refers to independent tasks that should be assigned to processors of various characteristics under the goal of minimizing the latest task's finish time. This value is known as makespan and is usually used as the performance metric for scheduling problems. Scheduling problems are in general NP-Hard [9] and this fact is commonly used as a justification that exact methods cannot be applied on solving them when problems of considerable size should be addressed.

* Corresponding author.
E-mail address: cgogos@teiep.gr (C. Gogos).

http://dx.doi.org/10.1016/j.future.2016.01.016
0167-739X/

Fortunately, this is not the case for all scheduling problems, since in several cases, problem specific idiosyncrasies can be exploited, pushing up the limit of problem sizes that can be solved using exact methods. When problem sizes that can be solved using exact methods equals or exceeds the size of problems with practical importance these methods tend to have a solution quality advantage over various approximate methods. This seems to be the case with HCSP and the method CPR that is proposed in Section 5 of this manuscript.

Makespan might be the most common objective used but it has to be mentioned that a number of other objectives also exist such as resource utilization, flow-time and matching proximity [10]. Resource utilization measures the degree of utilization of the resources. Flow-time is the sum of the finishing times of the tasks and can be seen as an indicator of the Quality of Service (QoS) of the system. Matching proximity measures the proximity of the schedule to the schedule that assigns each task to the processor that can execute it fastest. Each one of the aforementioned objectives might be optimized independently or a multiobjective problem might be formulated that targets the simultaneous optimization of more than one objectives [11,12]. Nevertheless, the majority of published work on task scheduling regards makespan as the most important performance metric and uses it apart from other metrics for comparisons among produced schedules.

This manuscript is organized as follows. The next section introduces the problem, presents related work and describes in detail the heuristics Sufferage and Penalty Based (PB) since they form the base for the novel heuristics LSufferage and TPB that are presented next. It is argued that LSufferage and TPB can be used as alternatives to already established heuristics for the problem. Next, a mathematical model of the problem and an associated Column Pricing approach is presented. The Column Pricing approach combined with some simple heuristics is capable to solve problem instances of large sizes that would have been impractical to address directly with an Integer Programming solver. The next section presents experiments that demonstrate the efficiency of the two new heuristics and the Column Pricing approach across a large number of known problem instances. Finally, conclusions of the research alongside with future directions are presented.

2. Problem description

The problem of assigning tasks to processors and specifying the order of execution on each processor is referred to as mapping. In the problem setting of this work tasks are considered to be independent, meaning that the completion of execution of a task is not a precondition for the execution of another task. The set of tasks is usually called meta-task and the goal is to minimize the makespan of the meta-task. Mapping of the meta-task occurs before the start of the meta-task execution, so the problem can be categorized as static scheduling. Furthermore, once a task is scheduled on a processor this task has to finish its execution on it, so pre-emptive execution is prohibited. A related problem that can be found in the bibliography is the Directed Acyclic Graph (DAG) scheduling problem. In contrast with HCSP, the DAG scheduling problem assumes a DAG that captures the precedence constraints and the communication costs that incur when a task that is scheduled to a processor has to transfer data to another task that is supposed to be scheduled to another processor [13]. Meta-tasks occur in several real life situations. For example a set of individual jobs submitted independently for execution to a grid computing is a meta-task.

2.1. The expected time to compute model

In order to simulate different Heterogeneous Computing environments and meta-task characteristics the Expected Time to Compute (ETC) model was used. ETC model was introduced by Ali et al. [14] and defines three metrics: task heterogeneity, processor heterogeneity and consistency type. Task heterogeneity refers to the variation among the execution times of tasks for a given processor. Processor heterogeneity refers to the variation of execution times for a single task across all the processors. Both of them can be either high or low. The third metric, consistency, assumes three values: consistent, inconsistent or partially consistent. When a dataset is consistent this means that if a processor executes a task faster than another processor, then it should execute all other tasks faster than the other processors. If the previous assertion does not hold, then the dataset is considered to be inconsistent except for the cases of semi-consistent datasets where a consistent subset of processors can be identified among them. The expected execution times of the tasks are arranged in a matrix called ETC, where the entry $ETC[t, p]$ is the expected time to compute task t on processor p assuming that the time needed to move the executables and the associated data is also included. These values are supposed to be known a priori while approaches that can be used in order to extract them include task profiling and analytical benchmarking. Further details about the ETC model alongside with alternative computational models for Grid scheduling can be consulted in [10]. Twelve instances of the ETC model involving 512 tasks and 16 processors were generated in [15] and since then used in several papers. Furthermore, 96 much larger problem instances of up to 8192 tasks and 256 processors, were introduced in [16]. So, a testbed of public problem instances and associated published results exist. This testbed was used in order to assert the performance of the approaches proposed later in this paper.

2.2. Related work

A non exhaustive list of approaches to HCSP that our research found includes heuristics [15,17], local search [18,19], Simulated Annealing [20], Genetic Algorithms (GAs) [21–23], Memetic Algorithms (MAs) [24,25], Ant Colony Optimization (ACO) [26,27] and Bee Colony Optimization (BCO) [28]. Among all approaches found, a parallel version of GA called pμ-CHC [16], running on a four QuadCore server cluster, reached very good makespan values close to the theoretical optimums for the same problem instances that were mentioned in the previous paragraph.

Heuristics might not give optimal results but are often used in practice due to their low computational cost and ease of implementation. Several of them are referenced in the bibliography [29] and one way to categorize them is according to the moment that they make the decision of scheduling. The resulting categories are online and batch mode. Online heuristics schedule tasks immediately upon arrival. For example MCT (Minimum Completion Time) heuristic assigns each task to the processor that will result to the earliest completion time for it. On the other hand, batch mode heuristics schedule each task by exploiting the execution times known about the other tasks. The following batch mode heuristics: Min–Min, Min–Max, RC (Relative Cost), PB (Penalty Based) and Sufferage, will be presented next. In all of the algorithms that follow, T is the set of tasks, P is the set of processors, $\text{ETC}(t, p)$ is the estimated time to complete task $t \in T$ on processor $p \in P$ and $\text{EFT}(t, p)$ is the earliest finish time of task t on processor p given the current state of assignments that have already been set up to the point of scheduling task t.

2.2.1. The Min–Min algorithm

Min–Min algorithm is a simple scheduling algorithm that is reported to consistently produce good results. In each step of Min–Min a single task is scheduled. This occurs by computing the earliest finish time (EFT) over all processors of all still unscheduled tasks in order to schedule the task to the processor that has the minimum EFT value over all other alternatives. Min–Min