Simulation Modelling Practice and Theory 19 (2011) 540–552

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

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

Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques

Georgios L. Stavrinides *, Helen D. Karatza

Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

ARTICLE INFO

Article history:
Received 4 July 2010
Received in revised form 18 August 2010
Accepted 22 August 2010
Available online 19 September 2010

Keywords:
Heterogeneous distributed real-time systems
Schedule holes
Task graphs
Bin packing
Performance evaluation

ABSTRACT

The most crucial aspect of distributed real-time systems is the scheduling algorithm, which must guarantee that every job in the system will meet its deadline. In this paper, we evaluate by simulation the performance of strategies for the dynamic scheduling of composite jobs in a heterogeneous distributed real-time system. Each job that arrives in the system is a directed acyclic graph of component tasks and has an end-to-end deadline. For each scheduling policy, we provide alternative versions which allow the insertion of tasks into idle time slots, using various bin packing techniques. The comparison study, based on different workloads and system heterogeneity levels, shows that the alternative versions of the algorithms outperform their respective counterparts.

1. Introduction

The inherent need of many scientific and commercial applications for completion within strict timing constraints, has led to the widespread use of distributed real-time systems. These systems are employed in various critical areas, such as the control of nuclear power plants, financial markets, radar systems, telecommunication networks, intensive care monitoring and multimedia applications. In a real-time system, the jobs have *deadlines* that must be met. The correctness of the system does not depend only on the logical results of the computations, but also on the time at which the results are produced. If a real-time job fails to meet its deadline, then its results will be useless, or even worse, this may lead to disastrous consequences for the system and the environment that is under control [2].

With the advent of increasingly powerful and cost-effective computing components, many of today's distributed real-time systems are heterogeneous. That is, they comprise different types of processors and networks. The major challenge in such a real-time heterogeneous environment, is the employment of effective scheduling techniques, in order to guarantee that every real-time job will finish its execution within the imposed timing constraints. The scheduling algorithm is responsible for the allocation of processors to jobs and determines the order in which jobs will be executed on processors.

In distributed real-time systems, a job may be represented by a *directed acyclic graph* (DAG or task graph), where the nodes represent the component tasks of the job and the edges represent the data dependencies between the tasks. Each job has an *end-to-end* deadline that must be met. That is, the tasks of a job do not have any specific individual deadlines, but there is an end-to-end timing constraint over all the tasks of the job. In order for a task to start execution, all of its

* Corresponding author. Tel.: +30 2310 997974.
E-mail addresses: gstavrin@csd.auth.gr (G.L. Stavrinides), karatza@csd.auth.gr (H.D. Karatza).

1569-190X/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2010.08.010

predecessor tasks must have been completed. A task with no predecessors is called an *entry* task, whereas a task with no successors is called an *exit* task. The immediate predecessors of a task are called *parents* of the particular task, while the immediate successors of a task are called *children* of the particular task. In order for a job to meet its deadline, i.e. to be *guaranteed*, all of its exit tasks must finish execution before the job's deadline is reached.

1.1. Related work

Because of its key importance, scheduling in distributed real-time systems has been studied extensively in the literature [3,8,9,15,22–24,18,20,21]. Among the real-time scheduling policies that have been proposed, the *Earliest Deadline First* (EDF) algorithm is the most commonly used [16]. According to this technique, the job with the earliest deadline has the highest priority for scheduling. Heuristics for task graph scheduling have been proposed by many authors [7,14,13,19,26]. Most of them are based on the *list scheduling* approach [11]. According to this technique, all ready tasks are arranged in a list in descending order of priorities. List scheduling consists of two phases: the task selection phase and the processor selection phase. In the first phase, the task with the highest priority is selected for scheduling, whereas in the second phase, the selected task is scheduled to a processor that minimizes a specific cost function, such as the estimated start time of the task.

One of the simplest list scheduling algorithms is the *Highest Level First with Estimated Times* (HLFET), or more simply, the *Highest Level First* (HLF) policy [1]. HLF assigns priorities to the tasks according to their position in the graph. More specifically, the task with the highest priority is the one with the highest *level*. The level of a task is the length of the longest path from that task to an exit task. Traditionally, HLF considers only task execution times when calculating levels. For the dynamic scheduling of multiple task graphs in multiprocessor real-time systems, Cheng et al. proposed in [4] a novel scheduling heuristic, called *Least Space–Time First* (LSTF), that takes into account both the precedence and the timing constraints among the tasks. However, the communication costs between the tasks in a DAG are not taken into account.

Kruatrachue and Lewis [10] proposed the *Insertion Scheduling Heuristic* (ISH), based on the observation that *schedule holes* (i.e. idle time slots) may form in the schedule, due to the communication delays between succeeding tasks in a DAG. ISH is an improved variant of HLF, where in the processor selection phase, a ready task may be inserted into a schedule hole on a particular processor, if it is able to execute within the particular idle time slot, but it is not able to start earlier on any other processor, either with exploitation of schedule holes or not. The task prioritization phase in ISH is exactly the same as in HLF.

In [25], Topcuoglu et al. investigate the scheduling of a single DAG in a heterogeneous distributed system. A static list scheduling policy is presented, the *Heterogeneous Earliest Finish Time* (HEFT) algorithm. HEFT prioritizes tasks according to their *upward rank*. The upward rank of a task is essentially its level, calculated based on the mean computational and the mean communication costs of the nodes and the edges respectively, on the path from the particular task to an exit task. The task with the highest upward rank has the highest priority for scheduling. In the processor selection phase, HEFT uses an insertion-based policy similar to ISH, that considers the possible insertion of a ready task into an earlier idle time slot between two already scheduled tasks on a processor.

1.2. Our approach

In this paper, we investigate the scheduling of multiple task graphs with end-to-end deadlines in a heterogeneous distributed real-time system, employing a list scheduling heuristic, where the task selection phase is based either on: (a) EDF, which takes into account only the deadlines of the jobs for the task prioritization, (b) HLF, which considers only the levels of the tasks, ignoring any timing constraints, or (c) LSTF, which assigns priorities to the tasks taking into account both the deadline of their job, as well as the level of each task. In our approach, the communication cost is taken into account in both HLF and LSTF. In the processor selection phase, we select the processor which can provide the selected ready task with the earliest estimated start time, either: (a) without the exploitation of possible schedule holes or (b) by exploiting idle time slots, as in the case of ISH and HEFT, but with the use of *bin packing* techniques.

The bin packing problem concerns the packing of a set of objects into a set of bins, using as few bins as possible. Some of the most popular bin packing policies are: *First Fit* (FF), *Best Fit* (BF) and *Worst Fit* (WF) [5,6,12,17]. FF is the simplest among these techniques. It places an object into the first bin where it fits. BF on the other hand, assigns an object to the bin where it leaves the minimum unused space possible. On the contrary, WF places an object into the bin where it leaves the maximum unused space possible. Essentially, the ISH and HEFT algorithms exploit possible idle time slots according to FF. In this paper however, the exploitation of possible schedule holes is based either on FF, BF or WF. To our knowledge, the exploitation of schedule holes with the explicit use of bin packing techniques has never been discussed in the literature before.

Our foremost goal is to guarantee that all jobs that arrive in the system will meet their deadlines. The performance of the scheduling policies is evaluated by simulation, under various workloads and system heterogeneity levels. The remainder of this paper is organized as follows: Section 2 gives a description of the system under study and the workload model, Section 3 describes the scheduling strategies, Section 4 explains how bin packing techniques are incorporated into the processor selection process in order to exploit schedule holes and Section 5 presents and analyzes the simulation results. Finally, Section 6 concludes this paper, providing suggestions for further research.