The Journal of Systems and Software 83 (2010) 1004–1014

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations

Georgios L. Stavrinides [*], Helen D. Karatza

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

ARTICLE INFO

Article history:
Received 2 October 2009
Received in revised form 13 December 2009
Accepted 21 December 2009
Available online 4 January 2010

Keywords:
Distributed real-time systems
Imprecise computations
Input error
Performance evaluation
Scheduling
Task graph

ABSTRACT

In order to meet the inherent need of real-time applications for high quality results within strict timing constraints, the employment of effective scheduling techniques is crucial in distributed real-time systems. In this paper, we evaluate by simulation the performance of strategies for the dynamic scheduling of composite jobs in a homogeneous 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 an alternative version which allows imprecise computations, taking into account the effects of input error on the processing time of the component tasks of a job. The simulation results show that the alternative versions of the algorithms outperform their respective counterparts. To our knowledge, an imprecise computations approach for the dynamic scheduling of multiple task graphs with end-to-end deadlines and input error has never been discussed in the literature before.

1. Introduction

Distributed real-time systems have become increasingly important in many domains of our daily life. Such systems are used for the control of nuclear power plants, financial markets, radar systems, telecommunication networks, medical 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 have disastrous consequences for the system and the environment that is under control (Buttazzo, 2004).

Consequently, in order to guarantee that every real-time job will produce high quality results within the imposed timing constraints, the employment of effective scheduling techniques is crucial in distributed real-time systems. The scheduling algorithm is responsible for the allocation of processors to jobs and determines the order in which jobs will be executed on processors.

Based on the observation that in real-time systems it is often more desirable for a job to produce an approximate result by its deadline, than to produce an exact result late, Lin et al. proposed the *imprecise computations* technique (Lin et al., 1987). According to this technique, the execution of a real-time job is allowed to return intermediate (*imprecise*) results of poorer, but still acceptable quality, when the deadline of the job cannot be met. To achieve this, it is assumed that every job is *monotone*. That is, the accuracy of its intermediate results is increased as more time is spent to produce them. If the execution of a monotone job is fully completed, then the results are precise (Liu et al., 1991).

A video-on-demand server, for example, which streams movies and other video content to users over the internet, may unexpectedly encounter network congestion, causing some packets to be lost or to be corrupted during the transmission of a video to the user. In this case, imprecise computations can allow the system to reduce the quality of some video frames during the transmission, so that the video delivered to the user maintains an acceptable frame rate. Another example, is the collision detection system in aircraft. This system collects data from the airplane's radar, identifies possible collisions with nearby aircraft and notifies the pilot. Imprecise computations can allow the system to ignore distant airplanes when an imminent collision is detected and therefore an immediate notification to the pilot is absolutely necessary. Hence, imprecise computations can provide scheduling flexibility in real-time systems, by trading off precision for timeliness (Feng, 1996; Hull, 2000; Shih and Liu, 1992).

In distributed real-time systems, jobs usually consist of component tasks with precedence constraints among them, so that a task's output may be used as input by other tasks of the job. Thus, jobs are *directed acyclic graphs* (DAGs) – i.e. *task graphs* – that have an *end-to-end deadline*. That is, component tasks do not have any specific individual deadlines, but there is an end-to-end timing

[*] Corresponding author.
E-mail addresses: gstavrin@csd.auth.gr (G.L. Stavrinides), karatza@csd.auth.gr (H.D. Karatza).

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.12.025

constraint over all the tasks of the job, that must be met. In order for a task to start execution, all of its 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 the case of imprecise computations, each component task is assumed to be monotone and that it consists of a *mandatory part*, followed by an *optional part* (Liu et al., 1994; Chung et al., 1990; Han et al., 2003). In order for a task to produce an acceptable result, its mandatory part must be completed. The optional part refines the result produced by the mandatory part. That is, the precision of a task's result is further increased, if the task's optional part is allowed to be executed longer. *Input error* may affect the processing time of some tasks. Specifically, if a task's result is imprecise, then the execution time of its child tasks may increase, since more time may be required by the child tasks to produce an acceptable result when there is error in their input (Hull et al., 1997). In order for a job to be completed, all of its tasks must complete *at least* their mandatory part before the job's deadline.

Scheduling in distributed real-time systems has been studied by many authors (Karatza, 2007, 2008; Stavrinides and Karatza, 2008, 2009; Seljak, 1994; Sha et al., 2004). Among the real-time scheduling policies that have been proposed in the literature, the *Earliest Deadline First* (EDF) algorithm is the most commonly used (Liu and Layland, 1973). According to this technique, the deadlines of the jobs are used for their priority assignment. Task graph scheduling strategies are examined in Kwok and Ahmad (1999); Iverson and Ozguner (1999) and Chen and Maheswaran (2002), such as the *Highest Level First* (HLF) policy (Adam et al., 1974), which assigns priorities to the tasks according to their position in the graph. For the dynamic scheduling of multiple task graphs in multiprocessor real-time systems, Cheng et al. (1997) proposed a novel scheduling algorithm, called *Least Space–Time First* (LSTF), that takes into account both the precedence and the timing constraints among the tasks. Feng and Liu (1997) investigate the impact of input error on the static scheduling of a single linear chain of tasks with an end-to-end deadline on a single processor. On the other hand, Haweet et al. (2003) examine the static scheduling of a single task graph with an end-to-end deadline on a single processor, but without taking into account the effects of input error on the execution time of the component tasks.

In this paper, we examine the performance of the EDF, HLF and LSTF algorithms for the scheduling of multiple task graphs (i.e. composite jobs with a directed acyclic graph structure) with end-to-end deadlines in a homogeneous distributed real-time system. We also provide an alternative version for each of the three algorithms, which allows imprecise computations, taking into account the effects of input error on the processing time of the component tasks of a job. Our foremost goal is to guarantee that all jobs that arrive in the system will meet their deadlines, providing high quality (precise) results. We compare the performance of the scheduling policies by simulation, under various workloads, based on the Overall System Performance metric, which is explained in Section 5. To our knowledge, an imprecise computations approach for the dynamic scheduling of multiple task graphs with end-to-end deadlines and input error has never been discussed in the literature before.

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 imprecise computations are incorporated into the scheduling process and Section 5 presents and analyzes the simulation results. Finally, Section 6 concludes this paper, providing suggestions for further research.

2. System and workload model

The target system is considered to be a homogeneous distributed real-time system, consisting of P processors, each serving its own queue (memory). Real-time jobs arrive in the system in a Poisson stream with rate λ. Each job J_k that arrives in the system, is a directed acyclic graph (DAG) of non-preemptible component tasks. Each vertex in the graph represents a task T_i of job J_k, whereas a directed edge E_{ij} between a task T_i and a task T_j represents a message that must be transmitted from task T_i to task T_j. A task can start execution only if it has received its required input data from all of its parent tasks. It is assumed that no additional data are required during a task's execution and that the output data of a task are available only after the task's completion.

Assigned to each edge (message) is a communication cost. The communication cost of a particular message between any pair of processors in the system is considered to be the same. Moreover, the communication cost between two precedence constrained tasks assigned to the same processor is considered to be negligible. It is assumed that the processors in the system are fully connected, that the communication is contention-free and that a processor can receive and send data to other processors simultaneously, even when it is executing a task. Thus, no attention is paid to the routing strategies used for communication. Hereafter, we will use the terms job, task graph and DAG interchangeably.

A job that arrives in the system may have one or more entry tasks and one or more exit tasks. Each component task T_i of a job J_k is characterized by the following parameters:

- Its service (execution) time S_i, which is exponential with mean $1/\mu$.
- Its priority value PV_i. The task with the smallest or largest priority value (depending on the scheduling algorithm which is employed) is considered to have the highest priority for scheduling.
- Its *level* L_i. The level of a task is the length of the longest path from the particular task to an exit task. The length of a path in the graph is the sum of the service times and the communication costs of all the tasks and edges on the path. The level of an exit task is equal to the task's service time.
- Its *rank* R_i, which denotes at which level of the graph J_k task T_i is located. For example, if T_i is an entry task, then its rank is $R_i = 1$, since it is located at the top level of the graph. The tasks that are located at the same graph level have the same rank and can be processed in parallel. A level in the graph should not be mistaken for the level L_i of a task T_i.
- The *set* $\mathscr{V}_i$ *of its parent tasks*.
- The *set* $\mathscr{Y}_i$ *of its child tasks*.

The communication cost C_{ij} of an edge E_{ij} between two tasks T_i and T_j in a job J_k is defined as follows:

$$C_{ij} = CCF_{ij} \cdot S_i \tag{1}$$

where CCF_{ij} is the *communication cost factor* of the edge E_{ij} and may be different for each edge in job J_k. S_i is the service time of the predecessor task T_i.

Consequently, a job J_k that arrives in the system is characterized by the following parameters:

- Its arrival time A_k.
- The number N_k of its component tasks. It is assumed that $1 \leqslant N_k \leqslant MAX$, where MAX is the maximum number of tasks in a graph.
- The *topological order list* $\mathscr{L}_k$ of its tasks. $\mathscr{L}_k$ is a list where all the tasks of the graph are placed in ascending order of their ranks and therefore, according to their topological order in the graph.