Adaptive Algorithms For Scheduling Static Task Graphs In Dynamic Distributed Systems

Prashanti Das¹, Dibyendu Das² and Pallab Dasgupta³

¹ Motorola India Electronics Ltd.
² Hewlett Packard, India Software Operation Pvt. Ltd.
³ Dept. of Computer Sc. and Engg, I.I.T Kharagpur, India

Abstract. In this paper we consider the problem of scheduling a given task graph on a dynamic network, where processors may become available or unavailable during the lifetime of the computation. We show that known list scheduling algorithms which use task cloning can be extended to develop efficient algorithms in this model. We also present a different approach where in anticipation of processor failures and recoveries, a set of schedules are precomputed and schedule switching is done whenever a failure or recovery takes place.

1 Introduction

Recent research on task scheduling with cloning has shown that cloning or duplicating tasks can significantly reduce the length of the overall schedule [2, 3, 4, 5]. While the task of finding an optimal schedule using cloning has been shown to be NP-hard, efficient polynomial time algorithms for task scheduling with cloning have been developed which give a good approximation of the optimal schedule [3, 4]. However, these algorithms assume that there is no bound on the number of processors. It has been shown that determining the processor requirement for optimally task scheduling with cloning is NP-hard [1]. Efficient heuristic based list scheduling algorithms have also been developed for task scheduling (with cloning) on a given set of processors [2, 5].

In this paper we address the following problem. We are given a directed acyclic task graph with known task execution times (represented as weights of the nodes of the task graph) and known data transfer times between pairs of tasks (represented as edge weights) if scheduled in different processors. The initial set of available processors is also given. With time, the set of available processors may change, that is, some processors may fail (become unavailable) and some processors may recover (become available). The set of available processors at a given instant of time becomes known only at that instant of time. We require fault tolerant strategies which reschedule and reallocate tasks dynamically and attempt to complete the computation represented by the given task graph at the earliest.

The contributions of this paper are as follows:

1. We extend the known heuristic based scheduling algorithms, ISH and DSH [5], to this model and show that the DSH-extension which uses cloning is well suited for this model.
2. We propose two strategies which pre-compute a set of non-dominated schedules, and perform a dynamic schedule mapping at runtime whenever a failure or recovery takes place. Experimental results show that the schedule lengths compare well with the DSH-extension, and therefore runs faster, since the scheduling overhead is minimized.

2 Problem Definition and Preliminaries

The algorithms presented in this paper are based on the following model. At a given instance of time, $\tau$, the distributed system consists of a set of homogeneous processors, $S^{\tau} = \{P_1^{\tau}, \ldots, P_k^{\tau}\}$. It is assumed that the distributed system is fully connected, that is, there is a communication channel between every pair of processors. Throughout this paper we ignore the contention on communication channels. The task scheduling problem in this model is defined as follows:

**Given:** A directed acyclic task graph consisting of the following:

1. A set of nodes representing a set of tasks $T = \{t_1, \ldots, t_n\}$ to be executed.
2. A partial order, $\prec$, defined on $T$ which specifies precedence between tasks. $t_i \prec t_j$ signifies that $t_i$ must be completed before $t_j$ can begin, that is, $t_j$ requires some data from $t_i$. The given task graph has an edge $(t_i, t_j)$ iff $t_i \prec t_j$.
3. For each task $t_i$, we are given its execution time, $w_i$. Since the system consists of similar processors, $w_i$ is independent of the processor in which it is executed.
4. For each edge $(t_i, t_j)$ in the task graph, we are given an edge weight, $c_{ij}$, which represents the time required to transfer data from $t_i$ to $t_j$ provided they are scheduled in different processors. $c_{ij}$ is proportional to the volume of data transferred from $t_i$ to $t_j$.

**To find:** The shortest length schedule for the set of tasks $T$ on the set, $S^{\tau}$, of available processors at time $\tau$.

The model for communication delays is as follows. We assume the existence of an independent communication interface for each pair of processors, that is, a processor can execute a task and communicate with one or more processors at the same time. In such a model, for any two tasks $t_i$ and $t_j$, if $t_i \prec t_j$ and $t_i$ has been scheduled in processor $P$ at time $\tau$, then $t_j$ should be scheduled either on processor $P$ at time $\tau' > \tau + w_i$ or on some other processor at time $\tau' > \tau + w_i + c_{ij}$. We make a few simplifying assumptions about the system:

- As soon as the set of available processors change, it becomes universally known and the rescheduling process can start without any delay.
- Each failure and each recovery occur in distinct instances of time.
- Each task, on completion, writes the data required by its successors on a shared (broadcast) medium. This is quite reasonable, since a shared filesystem (which acts as the broadcast medium) is common in clusters.

Based on these assumptions, we introduce the definitions of *edge dropping* and *node dropping*.