Scheduling Multi-Task Multi-Agent Systems

Rong Xie, Daniela Rus, Cliff Stein
Dartmouth College
Hanover, NH 03755
{rong,rus,cliff}@cs.dartmouth.edu

ABSTRACT

We present a centralized and a distributed algorithms for scheduling multi-task agents in heterogeneous networks. Our centralized algorithm has an upper bound on the overall completion time and is used as a module in the distributed algorithm. Extensive simulations show promising results.

1. INTRODUCTION

A *mobile agent system* is a single, unified framework in which a complicated distributed application (such as a complex query) can be easily implemented by writing a multi-task agent (also called the root agent), where there are possible precedence constraints and data transfers among the constituent tasks. At the choice of the programmers, each task can be carried out either by the root agent or by a child agent generated by the root agent.

A key component of any practical agent system is *agent scheduling*, i.e. controlling how the agents access resources. Such resources may include CPU time, memory, and disks, and can be provided by possibly more than one machine in the network. For scheduling a multi-task agent with the objective of minimizing its overall completion time, there is a tradeoff between the amount of utilized parallelism in the application and the amount of data transfer overhead incurred.

Compared with traditional computing environments, mobile agent systems are characterized by large data transfer delays, diversified network links and a wide spectrum of machine speeds. Therefore many assumptions used in traditional scheduling algorithms become unrealistic. Scheduling algorithms for a mobile agent system must work in a heterogeneous environment where (1) the number of machines is limited; (2) task graph structures are general; (3) data transfer delays are general; and (4) task duplication are generally not allowed. This problem is NP-Complete.

In this paper, we present a centralized and a distributed algorithm for scheduling multi-task agents in heterogeneous networks with the objective of minimizing the overall application completion time. The centralized algorithm has a provable performance upper bound and is used as a module in the distributed scheduler. Extensive simulations show promising results for the algorithms. The implementation of the algorithms in mobile agent systems is also discussed.

2. PROBLEM MODEL

We represent a distributed application as a *DAG (Directed Acyclic Graph)* $\mathbf{G} = (\mathbf{T}, \mathbf{E})$, where the set of nodes $\mathbf{T} = \{T_i\}_{i=1}^n$ corresponds to the set of tasks to be executed , and the set of weighted, directed edges $\mathbf{E}$ represents both the precedence constraints and the data transfers among tasks in $\mathbf{T}$. An edge $(T_i, T_j) \in \mathbf{E}$ implies that T_j can not start execution until T_i finishes and sends its result to T_j. In this case, we use $d(T_i, T_j)$ to denote the amount of data T_i sends to T_j. Let $\text{Pred}(T_i)$ denote the set of all the immediate predecessors of task T_i. Let $\mathbf{M} = \{M_k\}_{k=1}^m$ be the set of machines in a fully connected network. Let $r(M_l, M_k)$ be the data transfer rate between machine M_l and M_k, which is ∞ if $l = k$. The processing time of task T_i on machine M_k is denoted by $p(T_i, M_k)$, which is ∞ if T_i cannot be executed on M_k.

The objective of the scheduling problem is to find an assignment map $M : \mathbf{T} \to \mathbf{M}$ and a set of starting times $st(T_i), i = 1, \cdots, n$, where each task T_i is scheduled to be processed on machine $M(T_i)$ starting at time $st(T_i)$, such that the precedence constraints are satisfied and the schedule length $C_{\max}$ is minimized. Here $C_{\max}$ is defined by

$$C_{\max} \triangleq \max_{1 \le i \le n} ft(T_i) = \max_{1 \le i \le n} (st(T_i) + p(T_i, M(T_i))),$$

where $ft(T_i)$ is the finish time of T_i.

3. CENTRALIZED SCHEDULING

3.1 Basic scheduling

Inspired by [1], we propose a centralized algorithm (Algorithm 1), in which an agent consisting of n tasks is scheduled in n steps, one task at a time.

At each step l, task T_i is called *ready* if it is not scheduled yet and all of its predecessors have been scheduled. The *data available time* $DA(T_i, M_k)$ of a ready task T_i on machine M_k is the earliest time T_i can start execution on M_k:

$$DA(T_i, M_k) \triangleq \max_{T_j \in \text{Pred}(T_i)} \left[ft(T_j) + \frac{d(T_j, T_i)}{r(M(T_j), M_k)} \right].$$

The *machine available time* $MA(M_k)$ for each M_k is the time that the all the tasks assigned to M_k so far finish processing.

Algorithm 1 (Basic Algorithm)
Initialize the set of ready tasks R as the set of entry tasks in T. At each scheduling step l, do:

- Find ready task T_{i^*} and machine M_{k^*} minimizing
$$\max\{DA(T_i, M_k), MA(M_k)\} + \frac{c * p(T_i, M_k)}{\max_{1\leq j\leq m}\{p(T_i, M_j)\}} \quad (1)$$
among all pairs of ready tasks $T_i \in R$ and machines $M_k \in M$; Schedule task T_{i^*} on machine M_{k^*}.
- $l := l + 1$. Update R. Terminate if $R = \emptyset$.

3.2 Forward Backward (FB) scheduling

Algorithm 1 may perform poorly if the DAG contains bad in-trees (i.e. there are tasks expecting large data from at least two of their predecessors). To overcome this problem, we also work with the inverse of the DAG obtained by reversing the direction of all its edges. Then *the inverse DAG has the same minimal schedule length as the original DAG.*

Algorithm 2 (FB algorithm)

1. Run the basic algorithm (Algorithm 1) on the original DAG $\mathbf{G} = (\mathbf{T}, \mathbf{E})$, get schedule S;
2. Run Algorithm 1 on the inverse DAG $\hat{\mathbf{G}} = (\mathbf{T}, \hat{\mathbf{E}})$, reverse the generated schedule to get a schedule S'.
3. If $C_{max}(S) < C_{max}(S')$ output S, otherwise output S'.

3.3 PFB scheduling

In case the DAG contains both bad in-trees and bad out-trees (i.e. there are tasks sending large data to at least two of their successors), we propose the *partial forward backward* (PFB) algorithm.

Algorithm 3 (PFB algorithm)
Run Algorithm 1 on the original problem $\mathbf{G} = (\mathbf{T}, \mathbf{E})$ to get schedule S; Let $S' = S$. For each task $T_j \in T$, do:

- Reverse the part of schedule S' consisting of those tasks starting after time $\max_{T_i\in\text{Pred}(T_j)}(ft(T_i))$ in S' to get a partial schedule S_1, which is a schedule for those tasks in the inverse DAG; Starting from S_1, run Algorithm 1 on the inverse DAG for the remaining tasks to generate a complete schedule S_2 for the inverse DAG; Reverse S_2 to get a schedule S''.
- If $C_{max}(S'') < C_{max}(S')$, let $S' = S''$.

Output S'.

3.4 Performance analysis

Figure 1 shows the simulation results of FB and PFB algorithms versus the DLS algorithm presented in [1].

Theorem 1 *For scheduling problems with identical machines and heterogeneous communication links, the schedule length C_{max} generated by Algorithm 1, 2, or 3 satisfies*
$$C_{max} \leq (2 - \frac{1}{m}) * \bar{C}^*_{max} + D,$$
where $D = \sum_{i=1}^{K-1}\left[\frac{1}{m}\sum_{j=1}^{m} DA(B_i, M_j) - ft(B_{i+1})\right]$, and $L = B_K \rightarrow \cdots \rightarrow B_2 \rightarrow B_1$ is a special chain in G, and $\bar{C}^*_{max}$ is the optimal schedule length ignoring data transfers.

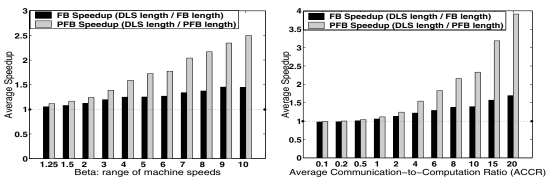

Figure 1: *Average speedup in C_{max} (over 100 simulation runs) with respect to machine heterogeneity and Average Communication-to-Computation Ratio (ACCR) respectively.*

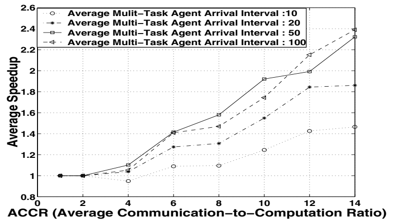

Figure 2: *Speedup in the sum of application turnaround time.*

4. DISTRIBUTED SCHEDULING

In a mobile agent system, multi-task agents arrive over time. We propose a distributed scheduling framework by assigning to each such agent its own scheduler (resource manager), which uses Algorithm 4 for scheduling. Here a task is called *ready* if all its predecessors have started executions.

Algorithm 4 (Distributed Algorithm)
Run Algorithm 3 on the multi-task agent to get its PFB schedule S_0; Initialize R as the set of entry tasks in $\mathbf{T}$.

While not all tasks of the agent have been scheduled, do:

- Update R; While $R \neq \emptyset$, do:
    1. Find the pair of ready task T_{i^*} and machine M_{k^*} that minimizes expression (1) among all pairs of ready tasks $T_i \in R$ and machines $M_k \in M$;
    2. If the Average Communication-to-Computation Ratio of the DAG is larger than λ, do: for each already scheduled task A that (1) has common successors with T_{i^*}; (2) is assigned on the same machine as T_{i^*} in S_0; (3) the minimum of the data transfer delays from A and T_{i^*} to their common successor is α times larger than the maximum of the standard execution times of A and T_{i^*}, set M_{k^*} as the machine that A is assigned to.
    3. Schedule task T_{i^*} on machine M_{k^*}.

Figure 2 shows the simulation results to compare the performances of Algorithm 4 using PFB hints (i.e. step 2 with $\lambda = 4$ and $\alpha = 2$) versus not using PFB hints.

Acknowledgment: This work bas been supported in part by Department of Defense contract MURI F49620-97-1-0382 and DARPA contract F30602-98-2-0107, ONR grant N00014-95-1-1204, NSF CAREER award IRI-9624286, Honda corporation, and the Sloan foundation; we are grateful for this support.

5. REFERENCES

[1] G.C. Sih and E.A. Lee. A compile-time scheduling heuristic for interconnection- constrained heterogeneous processor architectures. *IEEE Trans. on Paral. and Dist. Systems*, 4:175–186, 1993.