Scheduling Multi-Task Agents*

Rong Xie[1], Daniela Rus[1], and Cliff Stein[2]

[1] Department of Computer Science, Dartmouth College
{rong, rus}@cs.dartmouth.edu
[2] Department of IEOR, Columbia University
cliff@ieor.columbia.edu

Abstract. We present a centralized and a distributed algorithm for scheduling multi-task agents in a distributed system with the objective of minimizing the overall application completion time. Each agent consists of multiple tasks that can be executed on multiple machines which correspond to resources. The machine speeds and link transfer rates are heterogeneous. 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 of the algorithms, especially for scheduling communication-intensive multi-task agents.

1 Introduction

A *mobile agent system* is a single, unified framework for implementing distributed applications. Each distributed application can be implemented as a multi-task agent where there are possible precedence constraints and data transfers among the constituent tasks. The mobile agent executes by migrating from machine to machine, looking for data and resources according to each of its tasks.

A key component of any mobile agent system is controlling how the agents access the resources. Such resources may include CPU time, disk space, database access, etc. and may be provided by many machines in the network. For example, consider implementing a multi-step information retrieval as a multi-task mobile agent. The mobile agent will travel to a remote database to run a query with user-specified filters. The agent will then summarize locally the relevant results into a small number of topics or features. Using these features, the mobile agent will travel to a different database and register a persistent query, returning back to the user only after a set number of hits has been registered. If this database is replicated, the agent would have to choose which site to visit. The decision

* 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 N00014-01-1-0675, NSF CAREER award IRI-9624286, NSF award IIS 9912193, Honda corporation, and the Sloan foundation; we are grateful for this support. We are also grateful to Bob Gray and Susan McGrath for their efforts in developing the scheduling experiments using the D'Agents system. We thank Miguel Fernandez and Santiago Aja for their implementation of the scheduling system on top of the D'Agents.

depends on the general network traffic conditions, and the machine load and speed at the site of the database.

Since mobile agents move around in the network, often carrying variable size of data with them, the performance of an agent can be affected largely by data transfer delays, especially in heterogeneous networks with diversified network links. Thus, for scheduling a multi-task agent, there is a tradeoff between the amount of utilized parallelism in the agent and the amount of data transfer overhead incurred.

In this paper, we study the problem of scheduling multi-task agents in heterogeneous networks with the objective of optimizing the overall application completion time. Many assumptions used in traditional scheduling algorithms become unrealistic in this case. In general, scheduling algorithms for a mobile agent system must work in a heterogeneous environment where (1) the number of machines is limited; (2) precedence constraints are general; (3) data transfer delays are general; and (4) task duplication are not allowed. This problem is NP-Complete. In this paper, each agent consists of multiple tasks with precedence constraints, hence can be naturally modeled as a DAG (Direct Acyclic Graph). Both centralized and distributed scheduling algorithms are presented. In the centralized case, we present the FB and PFB algorithms which in a simplified case have a provable performance upper bound. In the distributed case, multi-task agents arrive over time. A distributed scheduling framework is proposed in which each multi-task agent is assigned its own scheduler which uses the PFB results as a module. Extensive simulations show promising results of the algorithms, especially for scheduling communication-intensive multi-task agents.

2 Problem Model

We represent each agent as a distributed application with a set of tasks among which there are possible precedence constraints and data transfers. This suggests using a DAG as representation. An instance of the agent (or, more generally the distributed application) is specified as a DAG $\mathbf{G} = (\mathcal{T}, \mathbf{E})$, where the set of nodes $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$ denotes the set of tasks to be executed and the set of weighted, directed edges $\mathbf{E}$ represents both precedence constraints and data transfers among tasks in $\mathcal{T}$. The existence of an edge $(T_i, T_j) \in \mathbf{E}$ implies 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 volume of data T_i sends to T_j. Let $\mathrm{Pred}(T_i)$ denote the set of all the immediate predecessors of task T_i.

Let $\mathbf{M} = \{M_1, M_2, \cdots, M_m\}$ be the set of machines across the network. We assume each pair of machines are connected to each other and $r(M_l, M_k)$ represents the data transfer rate between machine M_l and M_k. Since there is no communication delay for transferring data between two tasks on the same machine, we define $r(M_k, M_k)$ to be infinity. The processing time of task T_i on machine M_k is denoted by $p(T_i, M_k)$, which could be set to infinity if T_i can not be executed on M_k.

The objective of the scheduling problem is to find an assignment map $M : \{T_1, \cdots, T_n\} \to \{M_1, \cdots, M_m\}$ 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 the overall duration of the schedule defined as

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

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

Many approximation algorithms and heuristics have been proposed for DAG scheduling. Many of them assume the data transfer delay is negligible compared with the task execution time. For those considering data transfer delay, most of the results are purely empirical [10, 1, 3], or have various assumptions that do not hold for realistic applications, such as allowing task duplication to avoid long data transfer delays and assuming unlimited number of machines [8], restricting the structure of task graph [7], assuming globally small data transfer delay [6] or locally small data transfer delay [2].

3 Centralized Scheduling for a Multi-Task Agent

In this section, we propose two scheduling algorithms for a multi-task agent: the *forward-backward* (FB) dynamic priority algorithm and the *partial forward-backward* (PFB) dynamic priority algorithm. Both FB and PFB are based on a basic greedy algorithm illustrated in Section 3.1, though they can be combined with many other scheduling algorithms to enhance their performances as well.

3.1 Basic scheduling

In the basic algorithm, an agent consisting of n tasks in $\mathcal{T}$ is scheduled in n steps, one task at a time. Intuitively, if one task can start executing on one machine at the earliest time and with the fastest speed, we schedule this particular task on this particular machine. However, it is possible that by waiting a bit the task can be executed on a faster machine. Therefore, we select the best task-machine pair at each scheduling step by weighing two parameters: the time and speed at which one task can be executed. Fig. 1 presents our basic scheduling algorithm. This algorithm is inspired by the DLS algorithm presented in [10].

Let S_l be the system state at scheduling step l, which reflects the partial schedule information up to step l. S_l consists of the subset of T of all the tasks which have be scheduled before step l together with the machines they are assigned to and the scheduled starting times. At scheduling step l, task T_i is called *ready* if it is not scheduled yet and all of its predecessors have been scheduled. Let the set of all ready tasks be R.

At each scheduling step l, we define the *data available time* $DA(T_i, M_k)$ of a ready task $T_i \in R$ on machine M_k as the earliest time when all the data sent to task T_i from its predecessors is available at machine M_k:

$$DA(T_i, M_k) \triangleq \max_{T_j \in \mathrm{Pred}(T_i)} \left[ft(T_j) + \frac{d(T_j, T_i)}{r(M(T_j), M_k)} \right], T_i \in R,\ 1 \leq k \leq m.$$

In other words, $DA(T_i, M_k)$ reflects how soon all the data passed from T_i's predecessors can arrive at machine M_k. The *machine available time* $MA(M_k, S_l)$ for each machine M_k is the time when all the tasks assigned to M_k so far finish processing. $MA(M_k, S_l)$ is defined to be 0 if no task has been assigned to M_k.

Algorithm 1 (Basic Algorithm)

1. Initialization: Let the set of ready tasks R be the set of entry tasks in $\mathcal{T}$, *i.e.* those tasks with no predecessors;
2. At each scheduling step l, do:
   - For each pair of machine M_k and ready task T_i, where $T_i \in R, 1 \leq k \leq m$, compute its dynamic priority $DP(T_i, M_k, S_l) =$

$$\max\{DA(T_i, M_k), MA(M_k, S_l)\} + c * \frac{p(T_i, M_k)}{\max_{1 \leq j \leq m}\{p(T_i, M_j)\}} \quad (1)$$

   - Find the task-machine pair (T_{i^*}, M_{k^*}) such that $DP(T_{i^*}, M_{k^*}, S_l) = \min_{T_i \in R, 1 \leq k \leq m} DP(T_i, M_j, S_l)$
   - schedule task T_{i^*} on machine M_{k^*} after the last scheduled task on this machine.
   - Let $l = l + 1$. Update R and S_l.
   - Terminate if $R = \emptyset$.

Fig. 1. Basic algorithm

The max term in equation (1) (see Fig. 1) represents the earliest time task T_i can begin execution on machine M_k if T_i is scheduled on M_k. The second term reflects how fast task T_i can be executed on machine M_k. Since the execution time for one fixed task could be very different on different machines, we use this term to represent the relative efficiency of different machine-task combinations. The weight c is used to boost the weight of the second item in order to achieve a good compromise between these two criteria. The choice of c is currently experimental and deserves further study. In the case when two task-machine pairs have identical DP value, ties are broken by choosing the pair in which the task has a higher bottom-level, where the bottom-level of a task is defined as the largest sum of execution times along any path from this task to any exit task.

Notice that for each specific pair of ready task and machine , its DP value is different at different scheduling steps, and is nondecreasing with the increase of the scheduling steps. Hence algorithm 1 is a dynamic priority scheduling algorithm.

3.2 FB scheduling

There are situations in which the basic algorithm may generate very unsatisfactory schedules. Fig. 2 shows such an example.

The key structure in the agent's task DAG that causes performance degradation is the small triangle formed by tasks T_2, T_4, T_6, where T_6 requires large volume of data from T_2 and T_4, respectively. This is an important scenario, as it captures many mobile agent applications which perform information gathering and retrieval. Due to the greedy nature of the basic algorithm, when T_2 and T_4 are considered for scheduling, the scheduler only evaluates the quantities of data transferred to ready tasks T_2 and T_4, no consideration is given to the large data transfers from them to their common successors J_6. So the basic algorithm fails to assign T_2 and T_4 to the same machine, hence at least one of the two large data transfer delays must occur. In general, as long as the task graph contains structures where the data transfered to a single node from its multiple predecessors are all very large compared with task execution times, similar performance degradation will occur.

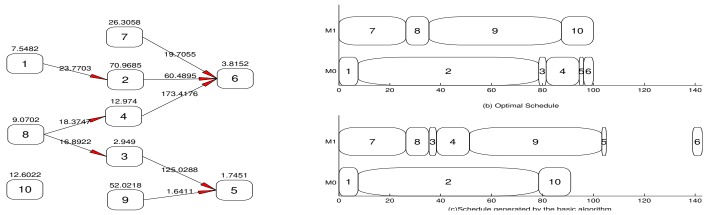


Fig. 2. *In this scenario we have two identical machines and the time needed to transfer d units of data between any two machines is d units of time. The weight of each node denotes the task execution time, and the weight of each edge denotes the volume of data to be transferred. The subgraph in* (*b*) *shows the optimal schedule, while subgraph in* (*c*) *shows the schedule generated by the basic algorithm, which is considerably longer than the optimal.*

To remedy this situation, we can enhance our scheduler by taking advantage of the forward-backward symmetry of the problem. Specifically, we define the inverse version of a given multi-task agent scheduling problem $\mathbf{G} = (\mathcal{T}, \mathbf{E})$ as $\hat{\mathbf{G}} = (\mathcal{T}, \hat{\mathbf{E}})$, where $\hat{\mathbf{E}} = \{(T_j, T_i) | (T_i, T_j) \in \mathbf{E}\}$. The task graph of the inverse problem is the same as the original one except the direction of each edge, *i.e.* the precedence relation, is inverted.

Proposition 1 *The inverse problem and the original problem have the same minimal makespan.*

Proof. Proof omitted for space considerations.

In the inverse problem, the data transferred from ready tasks in the original problem becomes data transferred to ready tasks, thus can be evaluated by the scheduler. This suggests that we can run Algorithm 1 on the inverse problem, then reverse the generated schedule (which is a feasible schedule for the inverse problem) to get a feasible schedule for the original problem. Fig. 4 summarizes this algorithm which we call *Forward Backward (FB)* dynamic priority scheduling. For the motivating example in Fig. 2, FB generates the optimal schedule shown in subgraph (c) of Fig. 3.

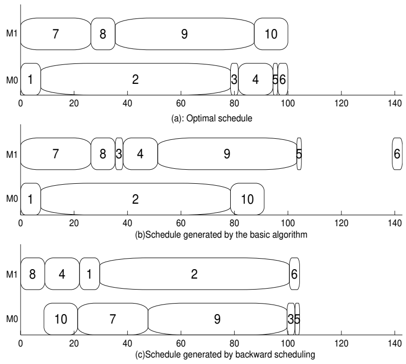


Fig. 3. A motivating example for extension

Algorithm 2 (FB algorithm)

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

Fig. 4. FB algorithm

3.3 PFB scheduling

Certain substructures in the multi-task agent's DAG enable the performance improvement of FB over the basic algorithm, particularly those "bad" in-tree structures where the data transferred to a single node from its multiple predecessors are all very large. By reversing the DAG, these in-tree structures will become "bad" out-trees (the data transferred from a single node to its multiple successors are all very large) and will be easily handled by the basic algorithm. However, when the DAG contains both bad in-trees and bad out-trees, the FB algorithm may fail to generate good schedules, since the forward or backward scheduling alone cannot handle both types of "bad" subgraphs simultaneously. Consider the example shown in Fig. 5. Both of the schedules generated by forward and backward scheduling suffer one long data transfer delay (100). The x-structure contains both the bad in-tree (ABC) and bad out-tree (CDE), which cause the considerable performance degradation. The bad in-trees and bad out-trees can also be independent of each other, as is shown in Fig. 6.

One natural solution is to use backward scheduling only on those parts of the DAG containing bad in-trees and forward scheduling on the remainder of the DAG, then assemble these two partial schedules together to get the final one. Partitioning the DAG optimally and efficiently is difficult. Fig. 7 shows our solution which we call the *partial forward backward (PFB)* dynamic priority scheduling algorithm.

For the example shown in Fig. 5, the partial backward scheduling is implemented by first reversing the part of the schedule for C, D and E generated by forward scheduling to get a partial schedule S_1 of the inverse DAG, in which C, D and E are scheduled on the first machine and start at time 2, 1, 0 respectively. Then tasks A and B are backward scheduled on the same machine as their predecessor C in the reversed DAG. Reversing the schedule for the inverse DAG, we get an optimal schedule for the original problem.

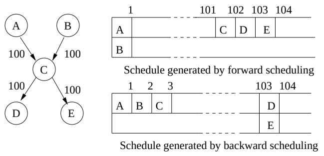


Fig. 5. A bad case for the FB algorithm.

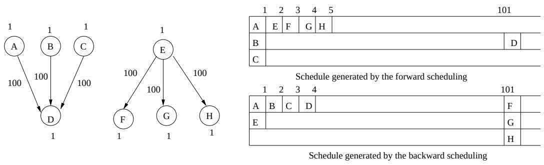


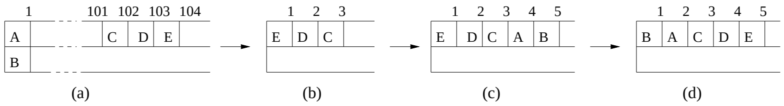
Fig. 6. Another bad case for the FB algorithm.

Algorithm 3 (*PFB* **algorithm**)

1. Run the basic algorithm (Algorithm 1) on original problem $\mathbf{G} = (\mathcal{T}, \mathbf{E})$, get schedule S;
2. Let $S' = S$; For each task $T_j \in T$, do:
   - Reverse the part of schedule S' consisting of those tasks starting after time $\max_{T_i \in \mathrm{Pred}(T_j)}(ft(T_i))$ 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 $S' = S''$.
3. Output S'.

Fig. 7. PFB algorithm

This scheduling process is demonstrated in Fig. 8.



(a) Schedule generated by forward scheduling; (b) Partial schedule of the inverse DAG for C, D and E; (c) Complete schedule of the inverse DAG; (d) Complete schedule of the original DAG.

Fig. 8. An scheduling example using PFB

3.4 Performance analysis

In this section, we present an upper bound for a simplified version of the basic algorithm in the computing environment in which machines are identical, but communication links differ.This is a salient feature of agent systems. In this situation, the second term in equation (1) becomes identical for all task-machine pairs, and thus can be ignored in evaluating the dynamic priority. The basic algorithm becomes the first-start-pair-first algorithm, *i.e.* the starting times of successively scheduled tasks are a non-decreasing sequence in time. Our basic algorithm generates the same schedule as the ETF algorithm in [4]. Our analysis is inspired by [4], but is much simpler.

We associate with each scheduling step q a time γ_q which is the DP value of the task-machine pair selected at that step, *i.e.* the starting time of the task scheduled at step q. We assume scheduling step q starts and completes instantly at time γ_q. Thus, saying that a task is ready-to-schedule at scheduling step q implies that a task is ready-to-schedule at time γ_q.

For scheduling problem $\mathbf{G} = (\mathcal{T}, \mathbf{E})$, let the schedule generated by the basic algorithm be S. As defined before, $st(B)$ and $ft(B)$ are the starting and finish time of task B in schedule S, respectively.

Lemma 1. *In schedule S, for every machine M_i and any task B such that $DA(B, M_i) < st(B)$, M_i is busy during the time interval $[DA(B, M_i), st(B)]$.*

Proof. Suppose otherwise M_i is idle during interval $[s, s+\Delta s] \subseteq [DA(B, M_i), st(B)]$ (see Fig. 9). Let C be the first task scheduled on M_i after $s + \Delta s$, then $C \neq B$ (otherwise the algorithm will schedule B at a time no later than s). Furthermore, task C must be scheduled before B, for if B is scheduled at a step p when C has not be scheduled, then $MA(M_i, S_p) \leq s$, which together with $DA(B, M_i) \leq s$ implies $DP(B,M_i, S_p) \leq s < st(B)$, a contradiction.

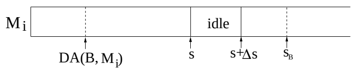


Fig. 9. Proof of Lemma 1

At the scheduling step q when C is being scheduled, B must be ready since it has not been scheduled and its data has been available since $DA(B, M_i) \leq$

$s < st(C) = \gamma_q$. Moreover, $MA(M_i, S_q) \leq s$, for M_i has been idle at least since time s. So $DP(B, M_i, S_q) = \max\{DA(B, M_i), MA(M_i, S_q)\} \leq s < st(C) = DP(C, M_i, S_q)$. Therefore B instead of C should be scheduled at this step, contradiction.

Let B_1 be the last task finishing in schedule S. Choose any chain $L = B_K \to \cdots \to B_2 \to B_1$ in $\mathbf{G}$ starting from some entry task B_K and ending at B_1. Denote the length of the schedule S as C_{max}, and the optimal schedule length ignoring data transfer delays as $\bar{C}^*_{max}$.

Theorem 1. *For scheduling problems with identical machines and general communication links,*

$$C_{max} \leq (2 - \frac{1}{m}) * \bar{C}^*_{max} + D, \tag{2}$$

where

$$D = \sum_{i=1}^{K-1} \left[\frac{1}{m} \sum_{j=1}^{m} DA(B_i, M_j) - ft(B_{i+1}) \right].$$

Proof. Define t_{idle} to be the sum of idle time on all machines before time C_{max} in schedule S. Similarly t_{busy} is the sum of busy time on all machines before time C_{max}. Hence $t_{idle} + t_{busy} = mC_{max}$. Since B_K has no predecessors, all machines must be busy before time $st(B_k)$, so

$$t_{idle} \leq (m-1) \sum_{i=1}^{K} p(B_i) + \sum_{i=1}^{K-1} \sum_{j=1}^{m} (st(B_i) - ft(B_{i+1})).$$

Since $\bar{C}^*_{max}$ is no smaller than the sum of execution time along any chain, and by Lemma 1, every machine M_j must be busy during the time interval $[DA(B_i, M_j), st(B_i)]$ if $DA(B_i, M_j) < st(B_i)$ for $i = 1, \ldots, K$, we have

$$t_{idle} \leq \ (m-1)\bar{C}^*_{max} + \sum_{i=1}^{K-1} \sum_{j=1}^{m} [DA(B_i, M_j) - ft(B_{i+1})] \tag{3}$$

Therefore, $C_{max} = \frac{1}{m}(t_{busy} + t_{idle}) \leq \bar{C}^*_{max} + \frac{1}{m} t_{idle}$, which together with (3) completes the proof of Theorem 1.

Among all the chains satisfying the conditions preceding Theorem 1, we can define a particular one $\tilde{L}$ as follows: Let $\tilde{B}_1 = B_1$. Fixing the starting times and the associated machines of all tasks in $Pred(\tilde{B}_1)$ as in schedule S, choose i_1^* such that $DA(\tilde{B}_1, M_{i_1^*}) = \max_{1 \leq i \leq m} DA(\tilde{B}_1, M_i)$, and let $\tilde{B}_2$ be one immediate predecessor of $\tilde{B}_1$ whose data for $\tilde{B}_1$ arrives last at machine $M_{i_1^*}$. So

$$f_{\tilde{B}_2} + d(\tilde{B}_2, \tilde{B}_1)/v(M(\tilde{B}_2), M_{i_1^*}) = DA(\tilde{B}_1, M_{i_1^*}).$$

Inductively define $\tilde{B}_i, i = 3, \cdots, \tilde{K}$ in this way, until reaching an entry task $\tilde{B}_{\tilde{K}}$. For this particular chain $\tilde{L}$, Theorem 1 becomes:

Corollary 1. *In equation (2), D can be written as*

$$D = \sum_{i=1}^{\tilde{K}-1} \frac{d(\tilde{B}_{i+1}, \tilde{B}_i)}{r(M(\tilde{B}_{i+1}), M_{i_i^*})} \leq \sum_{i=1}^{\tilde{K}-1} \frac{d(\tilde{B}_{i+1}, \tilde{B}_i)}{v_{min}}, \tag{4}$$

where v_{min} is the speed of the slowest link.

3.5 Experimental results

In this section, we present simulation results for our two multi-task agent scheduling algorithms and compare their performances with the DLS algorithm of [10]. DLS is one of the few task scheduling algorithms that supports general computation and data transfer delay in heterogeneous domains.

Our simulations are run on two sets of task graphs: random DAGs with predetermined optimal schedules proposed in [5] and random DAGs with unknown optimal schedules. We define $ACCR$ (Average Communication to Computation Ratio) of a distributed application as the average communication (data transfer) delay divided by average computation time of tasks. The parameter c in equation (1) is set to be 10.

Fig. 10 shows the comparison result of running simulations on random DAGs with predetermined optimal schedule length in homogeneous environment. Both of the FB and PFB algorithms generate considerably better schedules than DLS, especially for communication-intensive applications.

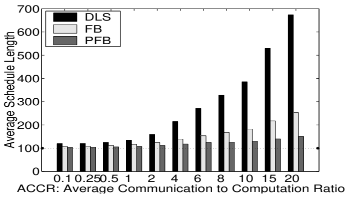


Fig. 10. *Random DAG with pre-determined optimal schedule length 100.0. The x-axis represents the $ACCR$; the y-axis represents the average schedule length (averaged over 60 simulation runs) color-coded for each of the three algorithms. Seven different values of $ACCR$ were selected: 0.1, 0.25, 0.5, 1, 2, 4, 6, 8, 10, to show the relative performance over a range of distributed applications from computation-intensive ones (when $ACCR$ is small) to communication-intensive ones (when $ACCR$ is large).*

Fig. 11 gives the average speedup when running simulations on random DAGs with unknown optimal schedules in *heterogeneous* environments. Here the speedup of algorithm A over algorithm B means the ratio of the schedule length generated by algorithm B to that generated by algorithm A. One hundred random DAGs were generated as test bed: the number of tasks in each DAG is of uniform distribution over $[20, 100]$, the average task execution times, average data transfer delays, machine speeds and link speeds are uniformly distributed over different ranges. The results shown in the graphs are an average over 100 separate simulations.

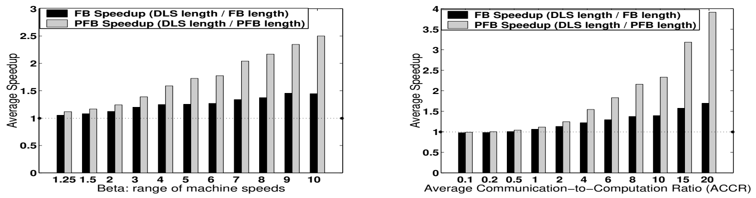


Fig. 11. *(a): Average speedup over DLS with respect to machine heterogeneity. The x-axis shows the range of machine speeds in the environment. The y-axis is the average speedup in the schedule length. Each bar is a value averaged over 100 simulations. (b): Average speedup with respect to ACCR. The x-axis is the ACCR and the y-axis is the average speedup in the schedule length. Each bar is a value averaged over 100 simulations.*

Fig. 11(a) shows the average speedup of our two algorithms over DLS with respect to machine heterogeneity, when the average task execution time, average data transfer delays were uniformly distributed over $[1.0, 9.0]$ and $[0.0, 40.0]$, respectively. The speed of each machine is of uniform distribution over the range $[\frac{1}{\beta} * p_i, \beta * p_i]$, *i.e.* a large value of β indicates a high heterogeneity of machine speeds. Seven different values of β ranging from 1.25 to 20 are selected to indicate different level of heterogeneity of machine speeds. The link rates vary uniformly over $[0.5*(\text{average rate}), 1.5*(\text{average rate})]$. Our algorithms outperforms DLS, and this performance improvement gets more evident as the range in which the machine speeds vary increases. We also observe a significant improvement of PFB over FB.

Fig. 11(b) shows the speedup of our algorithms over DLS with respect to $ACCR$, where the average task execution time, link rates and machine speeds are of uniform distribution over $[1.0, 9.0]$, [0.5$*$ average link rate, 2.0$*$ average link rate] and [0.2$*$ average machine speed, 5.0$*$ average machine speed], respectively. When $ACCR < 1$, DLS slightly outperforms our algorithms, but when $ACCR \geq 1$, our algorithms outperform DLS considerably. A significant improvement of PFB over FB is also observed.

4 Distributed scheduling for many multi-task systems

In a mobile agent system, multi-task agents arrive over time. In this section we use the ideas from scheduling a single multi-task agent to schedule many multi-task mobile agents in a distributed system. We propose a distributed scheduling framework by assigning to each such agent its own scheduler (resource manager), which uses Algorithm 4 for scheduling.

Algorithm 4 (Distributed Algorithm)

1. Run Algorithm 3 on the multi-task agent to get its PFB schedule S_0.
2. Initialize the set of ready tasks R as the set of entry tasks in $\mathcal{T}$, *i.e.* those tasks with no predecessors.
3. While not all tasks of the agent have been scheduled, do:
   - Update R;
   - While $R \neq \emptyset$, do:
     (a) For each pair of task T_i and machine M_k, where $T_i \in R$, $1 \leq k \leq m$, compute its dynamic priority from equation 1.
     (b) Find the task-machine pair (T_{i^*}, M_{k^*}) such that $DP(T_{i^*}, M_{k^*}) = \min_{T_i \in R, 1 \leq k \leq m} DP(T_i, M_j)$.
     (c) 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.
     (d) Schedule task T_{i^*} on machine M_{k^*}.

Fig. 12. Distributed scheduling

4.1 Model

We assume that each agent has its own scheduler, called the *agent scheduler*. We assume there is no communications between different agent schedulers, thus each agent scheduler works independently without cooperation. An agent scheduler takes a snapshot of the system state and makes scheduling decisions for the agent's tasks dynamically. Since multiple agents execute in the system, the actual starting time of a task may be different from the one computed by the agent scheduler. Thus, the notion of scheduling here is slightly different from what we have used in centralized scheduling, in that an agent scheduler does not specify the absolute starting time of each task.

For incoming agent tasks, each machine has two specific FIFO queues: a waiting queue and a ready queue, both of which are manipulated by a local "coordinator" agent residing on this machine. An incoming agent task is allocated to the ready queue or the waiting queue depending on whether its input data is available on this machine or not. For those tasks in the waiting queue, they can be reallocated to the ready queue by the coordinator agent at a later time when all its input data has arrived on this machine. Thus the tasks in the ready queue can start running instantly once the machine becomes idle, while the waiting queue consists of those tasks waiting for the arrivals of their input data. The coordinator agent is also responsible for notifying each agent scheduler when one of its tasks starts or finishes execution. By implementing these two queues on each machine, the tasks which are scheduled early but whose input data arrive very late will not block other tasks which are scheduled later but whose input data come earlier.

4.2 Distributed Scheduling

In a distributed mobile agent system, different multi-task agents arrive over time. Thus one factor that can affect the decisions of a scheduler is the time-varying machine states resulting from the arrival of tasks from other agents. Agent schedulers should dynamically, rather than statically, schedule their tasks to take into account the time-varying system states affected by the incoming tasks of other agents over time. An important issue in dynamic scheduling is timing, *i.e.* when to schedule the tasks of an agent. The scheduling time of a task can be as early as its agent's arrival time or as late as the time when the task is ready to run. If we schedule a task early, a large part of the scheduling and task submission overhead can be overlapped with the task computations and communications of the agent, but the state information can be stale. So there is a tradeoff between scheduling and task submission overhead, and the accuracy of the state information used by the scheduler. In our algorithm, we choose the scheduling time of a task to be the latest time among the starting times of all its predecessors, *i.e.* the earliest time when the data available times of this task on all machines can be calculated precisely. A task is ready to be scheduled when all its predecessors have started executions.

The centralized algorithms we developed in the previous sections can be extended to the distributed case. Algorithm 1 is adaptive, hence can be easily adopted by each agent scheduler. However, it can generate very unsatisfactory schedules when there are bad in-trees and bad out-trees in the multi-task agent DAG. On the other hand, the PFB algorithm can overcome this difficulty and improve the performance considerably, but its extension to the distributed environment is not straightforward. Therefore, it is natural to use the scheduling results of PFB algorithm as hints for the agent schedulers which utilize Algorithm 1 as the main scheduling scheme. The details of the overall distributed algorithm used by each agent scheduler are illustrated in Fig. 12, where λ and α are the threshold to determine whether the hints should be accepted or not, and the notations are the same as in Section 3. λ and α are usually above 1, since, from our previous simulations, the improvement by using PFB is significant only for communication intensive applications. The actual values of λ and α can be determined experimentally. This distributed extension only covers the bad in-tree structures in the DAG. which is of course not complete. However, in our previous simulations, we have observed that this is the main reason of performance degradation in scheduling problems with communication delays.

4.3 Simulation Results

Simulations are carried out in a heterogeneous computing environment, where the total number of machines is 16 and the machine speeds and link rates are generated randomly. There are 32 multi-task agents arriving over time, where the agent arrivals are given by a Poisson process. Each agent consists of 64 tasks, whose structure is generated randomly under the constraint that the maximum number of edges emitting from one task is 16.

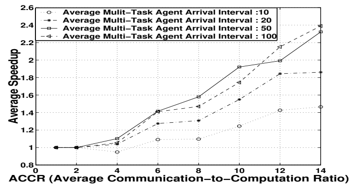


Fig. 13. *The performance of scheduling multi-task multi agent systems. The x-axis is the ACCR. The y-axis is the speedup in the sum of the application turnaround time. The four curves correspond to different average agent arrival intervals.*

In Fig. 13, we simulate our distributed scheduling algorithm to compare the performances of Algorithm 4 using PFB hints versus not using PFB hints. We choose the threshold λ and α to be 4 and 2 respectively. The results of each case are obtained by taking the average of five simulation runs. We observe that, by using PFB hints, the distributed algorithm has significant performance improvement when the ACCR is large, *i.e.* when scheduling multi-agent systems that are communication intensive.

The distributed algorithm assumes that the scheduler has full knowledge of the multi-task agent to be scheduled and the global information of the network, which is not realistic in most real systems. In many cases we are only able to feed the scheduler with the estimated values of parameters. So it is important to evaluate how tolerant the distributed scheduling algorithm is to the estimation errors of parameters. Three parameters are chosen to be tested individually: the standard task execution time, the size of data transferred among tasks, and the transfer rate of communication links. For each of them, three estimation error ranges are simulated, where the estimated parameters are uniformly distributed within $\pm 10\%, \pm 25\%, \pm 50\%$ of the correct values respectively. We define the degradation ratio as the ratio of the sum of the total application turnaround times using correct parameters to that using estimated parameters. We use this ratio to indicate the tolerance of our algorithm to parameter variations, where a degradation ratio far below 1.0 means that the algorithm is very sensitive to parameter variations. We evaluate the mean and the standard deviation of the degradation ratio under different average communication-to-computation ratios by averaging over twenty simulation runs. Fig. 14 and Fig. 15 show the simulation results. The average application arrival interval is 100. We observe that the algorithm is more sensitive to the data sizes and link rates than to the standard task execution times. The overall degradations are acceptable, where the worst-case performance degradation is 20%. When ACCR is large, we observe large standard deviations.

4.4 An Experiment

We are currently implementing our distributed mobile agent scheduling algorithms in the context of a multi-step information retrieval which is a component of the MURI application described in [9]. The scheduling algorithm has already been implemented on top of the D'Agents mobile agent system. The main components of the implementation consist of the scheduling modules used in our simulations and modules used to estimate network delay, machine load, and machine speed for each of the machines in the system. We hope to collect data consistent with our simulations for multi-step information retrievals in the near future.

5 Conclusions

We presented a solution to distributed multi-task multi-agent scheduling for mobile agent environments with heterogeneous hosts and communication delays. We approached this problem by first developing a centralized algorithm for scheduling a single multi-task agent. An upper bound is provided for this algorithm in a simplified case when all the machines in the system are identical but the communication delays vary. We then extend this algorithm for the distributed multi-agent problem, by associating a scheduler with each agent. Extensive simulation results show that the proposed algorithm is promising, especially for the distributed scheduling of communication-intensive multi-task agents.

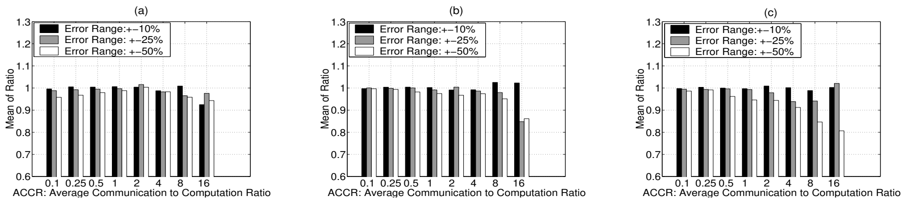


Fig. 14. *The x-axis represents the ACCR; the y-axis represents the mean of the degradation ratio. Tested parameter: (a)Standard task execution time; (b) Size of data transferred among tasks; (c) Transfer rates of communication links.*

References

1. H. El-Rewini and T.G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, 9:138–153, 1990.

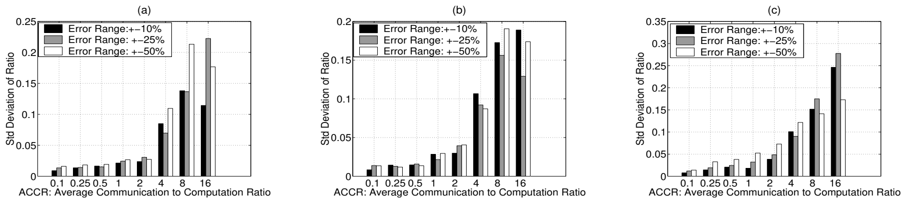


Fig. 15. *The x-axis represents the ACCR; the y-axis represents the standard deviation of the degradation ratio. Tested parameter: (a) Standard task execution time; (b) Size of data transferred among tasks; (c) Transfer rates of communication links.*

2. C. Hanen and A. Munier. *An approximation algorithm for scheduling dependent tasks on m processors with small communication delays.* 1995.
3. M. Iverson and F. Ozguner. Parallelizing existing applications in a distributed heterogeneous environments. In *Proc. of HCW*, pages 93–100, 1995.
4. J.J.Hwang and Y.C.Chow. Scheduling precedence graphs in system with interprocessor communication times. *SIAM Journal on Computing*, 18(2):244–257, 1989.
5. Y. Kwok and I. Ahmad. Benchmarking the task graph scheduling algorithms. In *Proceedigns of the first merged international parallel processing symposium and symposium on parallel and distributed processing*, pages 531–537, 1998.
6. R.H. Mohring, M.W.Schaffter, and A.S. Schulz. Scheduling jobs with communication delays: using infeasible solutions for approximation. In *Algorithms - ESA '96*, pages 76–90, Barcelona, Spain, 1996.
7. A. Munier. Approximation algorithms for scheduling trees with general communication delays. *Parallel Computing*, 25(1):41–48, 1999.
8. C.H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, 19(2):322–328, 1990.
9. R.Gray, G.Cybenko, D.Kotz, R.Peterson, and D.Rus. D'Agents: Applications and Performance of a Mobile-Agent System. *submitted to Software Practice and Experience*, 2000.
10. G.C. Sih and E.A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel and Distributed Sys.*, 4:175–186, 1993.