Master-Slave Tasking on Asymmetric Networks

Cyril Banino-Rokkones[1], Olivier Beaumont[2], and Lasse Natvig[1]

[1] Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
{Cyril.Banino, Lasse.Natvig}@idi.ntnu.no
[2] LaBRI, UMR CNRS 5800, Domaine Universitaire, 33405 Talence Cedex, France
Olivier.Beaumont@labri.fr

Abstract. This paper presents new techniques for master-slave tasking on tree-shaped networks with fully heterogeneous communication and processing resources. A large number of independent, equal-sized tasks are distributed from the master node to the slave nodes for processing and return of result files. The network links present bandwidth asymmetry, i.e. the send and receive bandwidths of a link may be different. The nodes can overlap computation with at most one send and one receive operation. A centralized algorithm that maximizes the platform throughput under static conditions is presented. Thereafter, we propose several distributed heuristics making scheduling decisions based on information estimated locally. Extensive simulations demonstrate that distributed heuristics are better suited to cope with dynamic environments, but also compete well with centralized heuristics in static environments.

1 Introduction

In this paper, we consider the allocation of a large number of independent equal-sized tasks onto a tree platform. We concentrate on tree-shaped platforms since they represent a natural framework for master slave tasking. More importantly, administrative organizations often rely on tree-shaped networks to interconnect computing resources [1]. Initially, the root of the tree (master node) holds a large bunch of tasks. Those tasks will be either processed by the master node or transmitted to its child nodes (also called slave nodes). Then, in turn, the child nodes face the same allocation problem (either processing the tasks locally or forwarding them to their child nodes). We consider the case where slave processors need to send back a file of results after processing each task. Even if this is the most natural situation, it is worth noting that most of the papers on independent tasks scheduling or Divisible Load Theory (DLT) do not consider those return communications. Targeted platforms are fully heterogeneous, i.e. both the processing resources and the communication resources have different capacities in terms of processing power and bandwidth. Moreover, the network links present bandwidth asymmetry in the sense that the bandwidth for sending tasks down the tree may be different from the bandwidth for returning results up the tree.

We concentrate on the influence of dynamic resource characteristics on the allocation scheme. In shared and unstable environments such as grids and peer to peer systems, the performance of the resources may well change during the execution of the whole process. In this context, it is not realistic to assume that one of the nodes knows at any time step the exact performance of all resources and is able to make optimal scheduling

W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 167–176, 2006.

decisions [1]. Therefore, the main question consists in determining whether the allocation scheme can make use of some static knowledge about the platform (for instance, the optimal solution computed from an initial snapshot of the platform), or whether we need to rely on fully dynamic scheduling schemes. In order to answer this question, we first derive optimal scheduling algorithms (with respect to throughput maximization). Then we present several heuristics. Some of them make their scheduling decisions using the optimal scheduling policy, computed using a snapshot of resource performance characteristics. Those heuristics may lead to optimal scheduling decisions in static environments. On the other hand, we propose a set of fully dynamic allocation heuristics that make their scheduling decisions only according to information measurable locally. Those heuristics may give poor results in static environments, but their performances are expected to be more robust in dynamic environments. We compare all those heuristics through extensive simulations using the SimGrid toolkit [2]. We rely on simulations rather than direct experiments in order to make a fair comparison between proposed heuristics. Indeed, simulation enables running of the different tests on computing platforms having exactly the same dynamic behavior. Moreover, SimGrid enables to define the trace of performance data over time for each processing or communication resource. Therefore, it is possible to compute (off-line) the optimal solution at any given time step and it is therefore possible to compare the performances of the different heuristics between them and against the optimal ideal solution.

The rest of the paper is organized as follows. Section 2 is devoted to a survey of related work, both DLT studies, independent tasks scheduling and on dynamic scheduling. Then, we present our platform model in Section 3 and how to find the optimal solution, in presence of return messages, in Section 4. Section 5 states the main Theorem of this paper, which provides a mean to optimize the nodes bandwidth utilization. Section 6 presents a task-flow control mechanism that regulates the amount of tasks and results buffered by the nodes throughout the execution. The set of centralized and distributed heuristics are described in Section 7. The methodology and results of the simulations are discussed in Section 8. Finally, we give some remarks and conclusions in Section 9. Due to space limitation, many of the technical details have been omitted, but can be found in the extended version of this paper [3].

2 Related Work

The problem of master-slave tasking on heterogeneous tree platforms has already been widely studied, both in the context of Divisible Load Theory (DLT) and independent tasks scheduling. A divisible load is a perfect parallel task that can be arbitrarily split and allocated to slave processors, without processing overhead. The overall load is first split at the master node in order to minimize the total execution time. Tasks are distributed in one round to the slaves, so that the master node makes the decisions about the set of slaves to be used, the amount of data to be sent to each slave, and the communication ordering [4,5,6]. When return messages are taken into account, two permutations must then be determined (one for tasks distribution and one for results collection) [7,8]. Although the complexity of this problem is still open, Rosenberg et al. [9] proved that in the case of a homogeneous single-level tree, the optimal schedule for both outgoing

and incoming messages can be determined, and the optimal LIFO and FIFO orderings are given in [10] for heterogeneous single-level trees.

On the other hand, when considering independent tasks scheduling, the master node faces the allocation problem for each task and the communications with its child nodes may well be split into several rounds [11,12,13]. Recently research studies have focused on steady-state scheduling, i.e. throughput maximization [11, 14, 15]. The steady-state scheduling approach has been pioneered by Bertsimas and Gamarnik [16] who considered packet routing and proposed to concentrate first on resource occupation rather than scheduling. The optimal solution for resource occupation, given link capacities, is obtained via a linear program. Then, an algorithm based on super-steps is proposed for building the actual schedule of packets. This idea has been adapted in [14] to the distribution of independent tasks on static platforms. Results collection was not considered in [14], but the linear program presented in Section 4 is a direct adaptation of the solution proposed in [14].

Dynamic scheduling of independent tasks has not been widely studied. Recently, Hong et al. [1, 15, 17] proposed a very nice algorithm, based on decentralized versions of flow algorithms. It is worth noting that this algorithm assumes a strongly different communication model than the one presented in this paper, and consequently cannot be easily adapted to our model. Here again, the results collection has not been considered.

3 Platform Model

The model considered in this paper is based on the model proposed in [14] that we augment by introducing communication weights for returning computation results back to the master. Processing nodes are assumed to be connected via a node-weighted edge-weighted tree $T = (V, E, w, c, c')$ as depicted in Figure 1.

Each node $P_i \in V$ represents a computing resource of weight w_i, meaning that node P_i requires w_i units of time to process one task. Each edge corresponds to a communicating resource and is weighted by two values: c_i which represents the time needed by a parent node to send one task to its child P_i, and c'_i which represents the time needed by the child P_i to send one result back to its parent. All the w_i's are assumed to be positive rational numbers since they represent node processing times. We disallow $w_i = 0$ since it would permit node P_i to perform an infinite number of tasks. Similarly, we assume that all c_i's and c'_i's are positive rational numbers since they correspond to the communication times between two processors. A node can perform three kinds of activity simultaneously: (i) it can process a task, (ii) it can receive a task file from its parent or a result file from one of its children, and (iii) it can send a result file to its parent or a task file to one

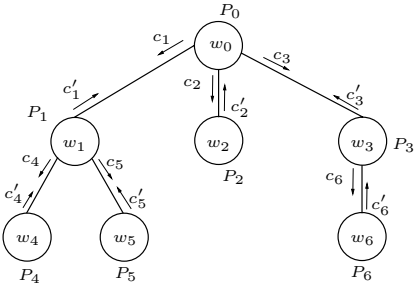


Fig. 1.

of its children. This model is known under the name *full overlap, bidirectional-single-port* model [14, 11]. At any given time-step, a node may overlap computation with only two connections, one for incoming communications and one for outgoing communications. Computation and communication are assumed to be atomic operations, i.e. once initiated they cannot be preempted. Finally the communication model works in a store-and-forward fashion.

4 Maximizing the Throughput

Given the resources of a weighted tree T operating under the *full overlap, bidirectional-single-port* model, we aim at maximizing the number of tasks processed per time unit. Let $\mathcal{C}_i$ denote the set of P_i's children. During one time unit, let α_i be the fractional number of tasks processed by P_i, and β_i be the fractional number of tasks received by P_i from its parent. Equivalently, α_i and β_i correspond respectively to the fractional number of results produced by P_i, and to the fractional number of results sent by P_i to its parent. The optimal throughput is obtained by solving the following linear programming problem (LPP), whose objective function is to maximize the number of tasks processed per time unit.

Maximize $n_{task}(T) = \sum_i \alpha_i$
subject to

$$\begin{cases} \forall i, & 0 \leq \alpha_i \leq \frac{1}{w_i} \\ \forall i \neq m, & 0 \leq \beta_i \\ \forall i \neq m, & \beta_i = \alpha_i + \sum_{j \in \mathcal{C}_i} \beta_j \\ \forall i, & \sum_{j \in \mathcal{C}_i} c_j \beta_j + c'_i \beta_i \leq 1 \\ \forall i, & \sum_{j \in \mathcal{C}_i} c'_j \beta_j + c_i \beta_i \leq 1 \end{cases}$$

The first set of constraints states that computation resources are limited. The second set of constraints confines the variables β_i within non-negative values. Note that the master P_m does not have a parent, so that we let $\beta_m = 0$. The third set of constraints deals with *conservation laws*. For each node P_i (except the master), the number of tasks received by P_i, should be equal to the number of tasks that P_i processes locally, plus the number of tasks forwarded to the children of P_i. Equivalently, the number of results sent by P_i to its parent, should be equal to the number of results produced locally by P_i, plus the number of results received from its children. The last constraints account for the single-port model. The send and receive operations performed by the nodes are assumed to be sequential.

Since we are looking for a solution of the LPP into rational numbers, optimal rational values for all variables can be obtained in polynomial time. However, the solution of the above LPP is in general not unique and some solutions might be more interesting than others in our context. In particular, *compact* solutions, i.e. that utilize nodes close to the root in priority, are more preferable than *stretched* solutions (that utilize nodes far away from the root). Indeed, start-up time (required to enter the steady-state) and wind-down time (required to gather the last results to the root) will be longer for stretched solutions than for compact ones. In order to obtain compact solutions, we first need to solve the initial LPP to derive the optimal throughput $n_{task}(T)$ of the tree. The objective function of the second LPP becomes the minimization of all the communications,

under the aforementioned constraints plus an additional one that states the conservation of the optimal throughput obtained by the former LPP. Minimizing the amount of communications while maintaining the optimal number of tasks processed implicitly enforces compact solutions. We hence add the following constraint: $\sum \alpha_i = n_{task}(T)$. And the objective function of the second LPP becomes: **Minimize** $\sum_i \beta_i$. Once a solution has been obtained, one needs to construct a schedule that (i) ensures that the optimal throughput is achieved and (ii) exhibits a correct orchestration of communication events, i.e. where simultaneous communications involve disjoint pairs of senders and receivers. We can obtain a time period Γ by taking the least common multiple (lcm) of all the denominators of the variables α_i. Then, the integer number of tasks γ_i that must be communicated to P_i during each time period Γ is obtained by $\gamma_i = \beta_i \Gamma$.

Proposition 1. *Sending and receiving files by bunches of γ_i in a round robin fashion generates an optimal steady-state schedule where single-port constraints are satisfied.*

Proof. The proof is done by induction over h, the height of the tree T [3]. □

Initially, nodes do not dispose of tasks nor results buffered locally to comply with Proposition 1. Therefore an initialization phase must take place before entering steady-state. During start-up, nodes will act as if they were in steady-state, at the difference that fake results will be sent to the parents if not enough results are available. Thus, tasks will be propagated down the tree, while fake results will be propagated up the tree. The fake results received by parents nodes are simply discarded. Once the first bunch of results processed by all the deepest nodes used in the schedule have been transmitted to the root node, then steady-state has been reached.

5 Bandwidth Optimization

A simple scheduling principle is presented in [14] when returning results is neglected. This scheduling algorithm was termed *bandwidth-centric* because priorities do not depend on the children processing capabilities, but only on their communication capabilities. The bandwidth-centric principle is extended to our problem as follows. First, observe that for each task that a node P_i delegates to a child P_j, P_i must first receive the task from its parent, then forward it to P_j, receive the associated result back, and finally send the result to its parent. Consequently, P_i will spend $x_j = c_j + c'_i$ time units sending data, and $y_j = c'_j + c_i$ time units receiving data. Since the master P_m does not have a parent, we let $x_m = c_m$ and $y_m = c'_m$. The bandwidth utilization of a node P_i can be sketched within the Cartesian plane, where the X and Y axes represent the time spent in emission and reception respectively. Hence, allocating a task to child P_j corresponds to a displacement in the Cartesian plane along vector $\boldsymbol{v_j}$ of components (x_j, y_j).

Theorem 1. *In steady-state, the bandwidth utilization of a parent node is optimized when using at most 2 children (if processing capabilities are not taken into account).*

Proof. The proof is done by induction over n, the number of children that are utilized by a parent in addition to the two nodes mentioned in Theorem 1. Consider the case where

$n = 1$, i.e. when a parent delegates α_1, α_2 and α_3 tasks per time unit to three children P_1, P_2 and P_3 respectively (see Figure 2). Displacements OA_1, A_1A_2 and A_2A_3 stand for delegating α_1, α_2 and α_3 tasks to the children P_1, P_2 and P_3 respectively.

Consider the triangle A_1A_2P where the displacements A_1P and PA_2 amount to allocate j_1 and j_3 tasks to P_1 and P_3 respectively. Consider now both quantities $(j_1 + j_3)$ and α_2. If $(j_1 + j_3) \geq \alpha_2$, it means that it is more profitable to spend the bandwidth time assigned to P_2 by allocating more tasks to P_1 and P_3. As a consequence, P_2 should not be used. But if $(j_1 + j_3) < \alpha_2$, then consider the triangle ORA_1, where the displacements OR and A_1R amount to allocate k_2 and k_3 tasks to P_2 and P_3 respectively. Since both triangles A_1A_2P and ORA_1 are equal (since their internal angles are equal), if $(j_1 + j_3) < \alpha_2$ then $(\alpha_1 + k_3) < k_2$. In that case, it becomes more profitable to assign k_2 tasks to P_2 instead of α_1 tasks to P_1 and j_3 tasks to P_3, and P_1 should not be used. Assume now that Theorem 1 is true for rank n, and let us prove that it holds also for rank $n + 1$. Consider a parent utilizing $n + 3$ children. Extract 3 of the $n + 3$ children and apply the aforementioned geometric transformation. One then utilizes only $n + 2$ children without degrading the initial throughput. □

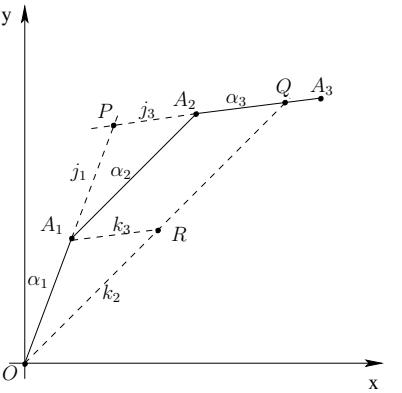


Fig. 2.

Theorem 1 assumes that nodes can provide as much computing power as necessary which contravenes the fact that computing resources are limited. Nonetheless, it allows identifying the way to optimize the bandwidth of any node P_i in using at most two children. Furthermore, we show in [3] that if such a pair of children exists, then the emission and reception bandwidth of P_i are equally utilized.

6 Task-Flow Control

In order to regulate the number of tasks and results that nodes are allowed to buffer locally throughout the execution, a threshold value θ_i is introduced for each node $P_i, i \neq m$. On the one hand, if the number of tasks buffered locally by P_i is beneath the threshold, then P_i will request more tasks in order to prevent starvation. On the other hand, if the number of results buffered locally by P_i is larger than the threshold, then P_i will not request additional tasks in order to hinder a monotonic accumulation of results. Initially, $\theta_i = 1, \forall i \neq m$. Since we search for compact solutions, parent nodes will try to process as many tasks as possible. If additional tasks arrive while a node is busy processing, then the task will be forwarded down the tree. During the execution, nodes are allowed to increase their local thresholds θ_i only when (i) they are starving and (ii) if they recently succeeded to accumulate θ_i tasks locally (to ensure that the current threshold is not sufficient) and (iii) if the number of results buffered locally is strictly lower than θ_i. This mechanism allows nodes to collect enough tasks locally to feed their

sub-trees, while ensuring that results do not accumulate monotonically locally. On the other hand, nodes must decrease their local thresholds whenever the number of results buffered locally exceeds the threshold. This threshold growth mechanism provides a mean to adapt to the platform dynamics.

7 Scheduling Heuristics

Round Robin. (RR) This heuristic implements Proposition 1. Once all the α_i are known, the period Γ is estimated as follows. Let us set $x = \lfloor \log_{10}(\max_i \alpha_i) \rfloor$. If $x \leq 0$ then $\Gamma = 10^{|x|+1}$, $\Gamma = 10^x$ otherwise. The aim is to obtain a compromise between a short time period, and an approximation close to the optimal solution. Then we get the number of tasks computed by each node P_i by rounding $\Gamma\alpha_i$ to the nearest integer.
On the Fly. (OTF) This heuristic makes use of the centralized knowledge. Once all the β_i's are known, each node maintains a table $tasks\_given[j]$, which records the number of tasks delegated to child P_j so far. The child node that has the lowest $\frac{tasks\_given[j]}{\beta_j}$ ratio is served in priority.
FIFO. Tasks are delegated in a first-come first-served basis.
Bandwidth-Centric. (BC) Let $r_j = \min\{\frac{1}{x_j}, \frac{1}{y_j}\}$ denote the maximum amount of tasks that P_i can delegate to child P_j per time unit. The child which has the highest r_j is served in priority.
Geometric. (Geo) This heuristic makes use of Theorem 1, but starts by applying the bandwidth-centric heuristic, in order to determine which child obtains the highest r_j. Then, it inspects if a pair of children can improve that rate. If such a pair of children exists, one must decide which child should be served. In order to make the right decision, we use a variable Δ which works much like a pair of scales. At start, $\Delta = 0$. Each time a child node P_j is served, we put x_j in one scale, and y_j in the other, which amounts to $\Delta = \Delta + x_j - y_j$. When a pair of children nodes is elected, then the child which brings Δ closest to 0 is serve. The aim is to utilize equally the emission and reception bandwidths of the parent nodes. Such strategy will optimize the bandwidth utilization of the nodes, while naturally adapting to the platform dynamics.

8 Simulations Results

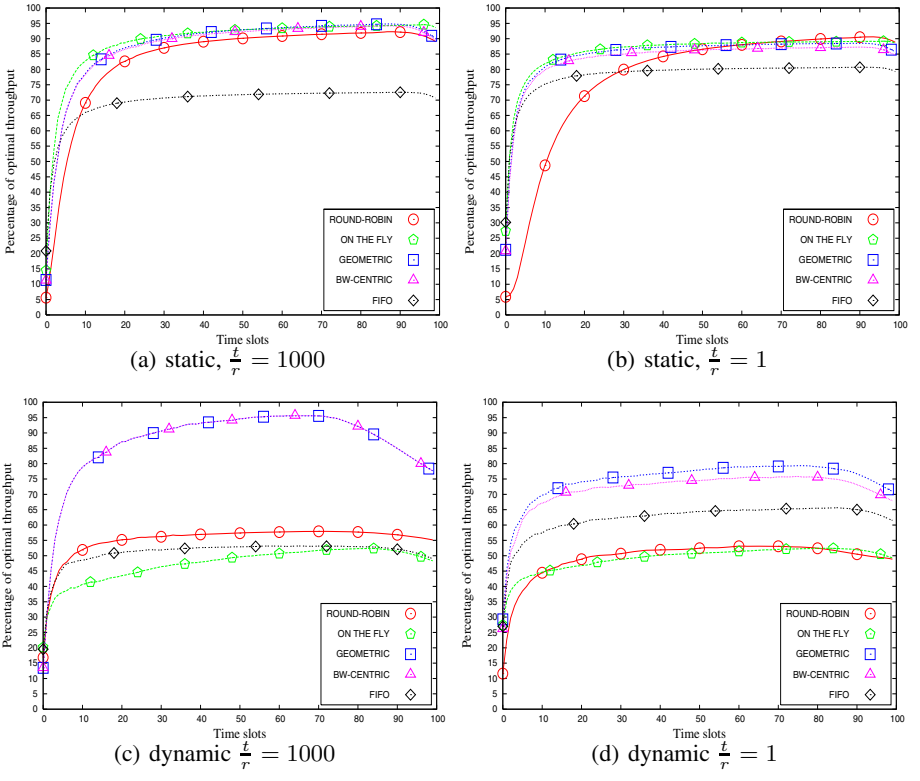
To evaluate our heuristics, we simulate the execution of an application on different random trees. Since a sub-tree can be reduced to a single super-node of equivalent processing power [14], it is not necessary to employ thousands of nodes to simulate large-scale systems [15]. We arbitrarily limited the number of nodes in a tree to 100. Each node was arbitrarily restricted to have at most 10 child nodes. A random tree is generated as follows. Each node is numbered with an ID number between 0 and 99. Then, each node $P_i, i \in [1, 99]$ is connected randomly to a node $P_j, j \in [0, i-1]$. The links have static performance values comprised between c_{min} and c_{max} and the nodes between w_{min} and w_{max}. All random distributions are uniform. The dynamic environments used in our simulations were generated as follows. Each resource R_i (node or link) has a cyclic behavior, i.e. its performance changes n_i times per cycle. The number of changes n_i

per cycle is randomly taken within the interval $[5, 15]$. Resource performance changes will occur every 25 treated tasks in average. We do not claim that these arbitrary decisions correspond to realistic network conditions. Our aim is to compare our heuristics on a set of different tree configurations. Inspired by Kreaseck et al. [11], we determine the throughput rate by using a growing window. The execution time is divided into 100 equal-sized time slots. Then, the window increases in size by step of one time slot, and the throughput rate delivered within the window time-frame is computed. The throughput rates delivered by the trees have been normalized to the maximum steady-state rates obtained with the LPP in static environments. However, throughput rates obtained in dynamic environments have been scaled up by a *dynamic factor* that accounts for the performance loss incurred by the platform dynamics. The dynamic factors have been obtained by successively solving LPPs of static platforms and comparing them to their homologous LPPs where some dynamism have been introduced (i.e. with the same platform topologies but with scaling down resource performances). More details about our methodology as well as a broader set of simulation can be found in [3].

In this paper, we report the simulation of an independent-task application of 2500 tasks on 50 trees where $c_{min} = 1$, $c_{max} = 10$, $w_{min} = 20$ and $w_{max} = 200$. Two scenarios for the data volume associated to the tasks and results were considered: (i) task data are 1000 times larger than result ones ($\frac{t}{r} = 1000$), and (ii) task and result data have the same size ($\frac{t}{r} = 1$). Figure 3 plots an average of the 50 throughput rates (associated to the 50 trees) over time. Figure 3 (a) and (b) correspond to static environments, while Figure 3 (c) and (d) correspond to fully dynamic environments, i.e. where resource performances can degrade down to 1% of the static value. The RR heuristic has been simulated with more than 2500 tasks in order to overcome the long start-up time required to enter steady-state. Still, RR does not outperform the other heuristics in static environments, certainly due to the truncating and rounding operations that occurred when computing Γ and the γ_i's. Not only the integer number of tasks intended to each node may be sub-optimal, but also the schedule of communications gets disturbed. The centralized heuristics (RR and OTF) are the highest performers in static environments, but the lowest ones in dynamic environments. Indeed, the information on which they rely throughout the execution becomes misleading in dynamic settings. As expected, the BC heuristic works very well when result data are small, while Geo only departs from BC when result data become significant.

Interestingly, when result data become significant, the performance of the best heuristics decrease, whereas the performance of FIFO increases. On the one hand, the decline of the best heuristics can be explained by the scheduling problem becoming more complicated. Returning results up the tree taking as long as sending tasks down the tree, parent nodes may sometimes have to stall a long time, waiting for a child to become available in reception. On the other hand, the performance increase of FIFO is a direct consequence of the task-flow control mechanism. When returning results takes a long time, local accumulations of results will arise, hindering the ineffective nodes to request for additional tasks. In contrast, when returning results is quick, no local results accumulations take place, increasing the margin to make wrong scheduling decisions.

Finally, it is worth noticing that BC and Geo compete well with the centralized heuristics even in static environments. See [3] for further details and interpretations.



(a) static, $\frac{t}{r} = 1000$ (b) static, $\frac{t}{r} = 1$

(c) dynamic $\frac{t}{r} = 1000$ (d) dynamic $\frac{t}{r} = 1$

Fig. 3. Average of the 50 throughput rates (associated to the 50 trees) over time, with the computation to communication ratio $\frac{w_i}{c_i} = 20$. In the dynamic environments, resource performances can degrade arbitrarily without failing, i.e. down to 1% of the static performance value.

9 Conclusion and Future Work

The problem of distributing a large number of independent tasks onto heterogeneous tree-shaped platforms with bandwidth asymmetry was considered. In contrast with most previous studies, the cost of returning results to the master node was represented in the problem formulation. We provided theoretical results that were embedded into autonomous heuristics. Simulations results showed that the autonomous heuristics put together with the task-flow control mechanism not only behaved very well in dynamic environments, but also compete well with centralized heuristics in static environments.

The scope of this paper was restricted to tree-shaped networks. However, at the backbone level, various geographically organizations are connected via the Internet resulting in a graph topology. Adapting the theoretical results presented in this paper to graph-shape platforms is a natural continuation of this work, albeit graph topology introduces routing problems. Another direction is to consider master-slave tasking in the presence of multiple masters. This situation arises naturally when several applications share the same platform, or when multiple masters collaborate on a single application.

References

1. Hong, B., Prasanna, V.K.: Performance Optimization of a De-centralized Task Allocation protocol via bandwidth and buffer management. In: CLADE. (2004) 108
2. Casanova, H.: SimGrid: A Toolkit for the Simulation of Application Scheduling. In: Proceedings of the 1st International Symposium on Cluster Computing and the Grid, IEEE Computer Society (2001) 430
3. Banino-Rokkones, C., Beaumont, O., Natvig, L.: Master-Slave Tasking on Asymmetric Tree-Shaped Networks. Technical Report 02/06, NTNU (2006) URL: http://www.idi.ntnu.no/∼banino/research/research.html.
4. Robertazzi, T.: Processor Equivalence for a Linear Daisy Chain of Load Sharing Processors. IEEE Trans. Aerospace and Electronic Systems **29** (1993) 1216–1221
5. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling Divisible Loads in Parallel and Distributed Systems. IEEE Computer Society Press (1996)
6. Drozdowski, M., Wolniewicz, P.: Experiments with scheduling divisible tasks in clusters of workstations. In: Proceedings of Euro-Par 2000: Parallel Processing. LNCS 1900, Springer (2000) 311–319
7. Barlas, G.D.: Collection-Aware Optimum Sequencing of Operations and Closed-Form Solutions for the Distribution of a Divisible Load on Arbitrary Processor Trees. IEEE Trans. Parallel Distrib. Syst. **9**(5) (1998) 429–441
8. Blazewicz, J., Drozdowski, M., Guinand, F., Trystram, D.: Scheduling a Divisible Task in a Two-dimensional Toroidal Mesh. In: Proceedings of the third international conference on Graphs and optimization, Amsterdam, The Netherlands, Elsevier Science Publishers B. V. (1999) 35–50
9. Adler, M., Gong, Y., Rosenberg, A.L.: Optimal Sharing of Bags of Tasks in Heterogeneous Clusters. In: 15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'03), ACM Press (2003) 1–10
10. Beaumont, O., Marchal, L., Robert, Y.: Scheduling Divisible Loads with Return Messages on Heterogeneous Master-Worker Platforms. In: International Conference on High Performance Computing HiPC'2005. LNCS, Springer Verlag (2005) 123–132
11. Kreaseck, B., Carter, L., Casanova, H., Ferrante, J.: Autonomous Protocols for Bandwidth-Centric Scheduling of Independent-Task Applications. In: IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, Washington, DC, USA, IEEE Computer Society (2003) 26.1
12. Dutot, P.F.: Complexity of Master-slave Tasking on Heterogeneous Trees. European Journal on Operationnal Research **164**(3) (2005) 690–695
13. Rosenberg, A.L.: Sharing Partitionable Workloads in Heterogeneous NOWs: Greedier is not Better. In: Cluster Computing 2001, IEEE Computer Society Press (2001) 124–131
14. Banino, C., Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Robert, Y.: Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms. IEEE Transactions on Parallel and Distributed Systems **15**(4) (2004) 319–330
15. Hong, B., Prasanna, V.K.: Distributed Adaptive Task Allocation in Heterogeneous Computing Environments to Maximize Throughput. In: International Parallel and Distributed Processing Symposium IPDPS'2004, IEEE Computer Society Press (2004) 52b
16. Bertsimas, D., Gamarnik, D.: Asymptotically optimal algorithm for job shop scheduling and packet routing. Journal of Algorithms **33**(2) (1999) 296–318
17. Hong, B., Prasanna, V.K.: Bandwidth-Aware Resource Allocation for Heterogeneous Computing Systems to Maximize Throughput. In: ICPP. (2003) 539–546