# Bandwidth-centric allocation of independent tasks on heterogeneous platforms

O. Beaumont[+], L. Carter[++], J. Ferrante[++], A. Legrand[+] and Y. Robert[+]

(+) LIP, UMR CNRS–ENS Lyon–INRIA 5668
Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
e-mail: Firstname.Lastname@ens-lyon.fr
(++) Computer Science Department, University of California, San Diego
9500 Gilman Drive, La Jolla, California 92093-0114 U.S.A.
{carter,ferrante}@cs.ucsd.edu

## Abstract

*In this paper, we consider the problem of allocating a large number of independent, equal-sized tasks to a heterogeneous "grid" computing platform. We use a tree to model a grid, where resources can have different speeds of computation and communication, as well as different overlap capabilities. We define a base model, and show how to determine the maximum steady-state throughput of a node in the base model, assuming we already know the throughput of the subtrees rooted at the node's children. Thus, a bottom-up traversal of the tree determines the rate at which tasks can be processed in the full tree. The best allocation is* bandwidth-centric*: if enough bandwidth is available, then all nodes are kept busy; if bandwidth is limited, then tasks should be allocated only to the children which have sufficiently small communication times, regardless of their computation power.*

## 1 Introduction

Mapping and scheduling tasks onto heterogeneous platforms has received considerable attention in recent years (see the survey papers of Berman [2] and Feitelson [5]). In this paper, we deal with the problem of allocating a large number of independent, equal-sized tasks to a heterogeneous "grid" computing platform. We allow a quite general model of computation, where the various resources that make up the grid can have not only different computation and communication speeds, but can even have different capabilities in terms of how much various communication tasks can be overlapped with computation. Our model is motivated by problems that are addressed by collaborative computing efforts such as SETI@home [8], factoring large numbers [3], the Mersenne prime search [7], and those distributed computing problems organized by companies such as Entropia [4].

We model a collection of heterogeneous resources and the communication links between them as the nodes and edges of an undirected graph. We assume that between any pair of nodes, there is only one path, that is, the graph is acyclic. Each node is a computing resource (a processor, or a cluster, or whatever) capable of computing and/or communicating with its neighbors at (possibly different) rates. To accurately determine the bandwidth experienced by an application, a tool such as the Network Weather Service [9] or Remos [6] can be used.

Given $n$ independent, identical tasks to be allocated on the grid, we assume that their data is initially located on (or generated by) a single node. We make this node the root of the graph, which, being acyclic, is now a tree. The root processor decides which tasks to execute itself, and how many tasks to forward to each of its children. Each child faces in turn the same dilemma: determine how many tasks to execute, and how many to delegate to each child. Due to heterogeneity, the children may receive different amounts of work.

In this paper, we show how each node *locally* can attain the best allocation of tasks to resources that maximizes the steady-state throughput, or tasks processed per unit time, throughout the tree. We also show that this best allocation is *bandwidth-centric*: if enough bandwidth is available to the node, then all children are

kept busy; if bandwidth is limited, then tasks should be allocated only to children which have sufficiently fast communication times, in order of fastest communication time. Counter-intuitively, the maximum throughput in the tree is achieved by delegating tasks to children as quickly as possible, and not by seeking their fastest solution.

In Section 2 we define our base model of communication and computation. We give in Section 3 a simple algorithm that finds the optimal allocation of tasks in this base model. We give some remarks and conclusions in Section 4.

The extended version of this paper [1] introduces several other models of communication and computation, and elaborates on the reduction of the allocation problem in any of the other models to a problem in the base model. Furthermore, we present in [1] simulation results of several demand-driven task allocation policies that show that our bandwidth-centric method obtains better results than allocating tasks to all processors on a first-come, first serve basis. Finally, we refer the reader to [1] for a comprehensive discussion of related work.

## 2 The base model

The target architectural/application framework is represented by a node-weighted edge-weighted tree $T = (V, E, w, c)$ as illustrated in Figure 1. Each node $P_i \epsilon V$ represents a computing resource of weight $w_i$, meaning that node $P_i$ requires $w_i$ units of time to execute one task. Each edge $e : P_i \rightarrow P_j$ is labeled by a value $c_{ij}$ which represents the time needed by node $P_i$ (the one closer to the root) to communicate the data for one task to node $P_j$ (one of its children) plus the time for the child to return the result when it is finished. For the purpose of computing steady-state behavior, it does not matter what fraction of the communication time is spent sending a problem and what fraction is spent receiving the results. To simplify the exposition, we will henceforth assume that all the time is spent sending the task's data to the child, and no time is needed to communicate the results back.

The only assumptions on the values $w_i$ and $c_{ij}$ we make are that they are integers with $w_i > 0$ and $c_{ij} \geq 0$. We disallow $w_i = 0$, since it would permit a node to perform an infinite number of tasks, and we restrict ourselves to integers (or equivalently to rational numbers) since with arbitrary real numbers, it could happen that no periodic could be within an additive constant of the optimal schedule asymptotically.

There are several scenarios for the operation of the processors, which are surveyed in [1]. Due to the lack

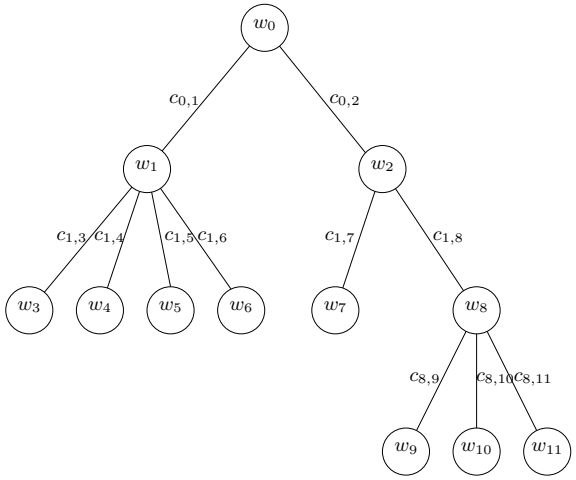


**Figure 1. A tree labeled with node (computation) and edge (communication) weights.**

of space, we concentrate on the *full overlap, single-port* model, where a processor node can simultaneously receive data from its parent, perform some (independent) computation, and send data to one of its children. At any given time-step, there are at most two communications taking place, one from the parent and/or one to a single child. This model is representative of a large class of modern machines, and is called the *base model*.

There are $n$ independent tasks to be executed on the tree. Each task requires $w_i$ units of time when executed by node $P_i$. So the smaller $w_i$, the faster the processor node $P_i$.

Initially, the data for all tasks is stored in the root processor. It takes $c_{ij}$ units of time for node $P_i$ to send (the data for) one task to its child $P_j$. In other words, if a task is resident on node $P_i$ at time-step $t$, and if $P_i$ initiates the communication to $P_j$ at that time-step, then the task will be available to $P_j$ at time $t + c_{ij}$. Tasks are atomic, their computation or communication cannot be preempted. A task represents the granularity of the application.

Given a weighted tree, one might want to determine how many tasks should be executed by each node so that the overall computation time is minimized. This is a difficult problem, and in fact cannot be answered without knowing more about the network than is in our model, such as the network latencies. Instead, we will answer a related question. As is well-known, a computation consists of a *start-up* or *initialization* interval where some processors are not running at the full speed that can be sustained, then a *steady-state* interval where all processors are running at the maximum speed that the network can sustain, and a *clean-up* interval

when some but not all processors are finished. During the steady-state interval, the operation of the tree is periodic, with $b$ tasks executed every $t$ time units. Both the start-up and clean-up times are bounded, no matter how many tasks there are. The question we will answer is what is the rate or *throughput*, $R = b/t$, of processing tasks in the steady-state interval. This answer can be used to give the execution time of the full application within an additive constant.

In steady state, the tree operates as a single (virtual) node whose computational weight $w_{\text{tree}} = \frac{t}{b} = \frac{1}{R}$ is to be minimized. The major contribution of this paper is to determine the best allocation of tasks to processors, according to the criterion of minimizing $w_{\text{tree}}$, for any input tree.

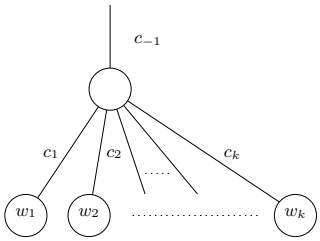


**Figure 2. Fork graph.**

## 3 Solution for the base model

### 3.1 Fork graph

We start with simple fork graphs before dealing with arbitrary tree graphs. A fork graph, as shown in Figure 2, consists of a node $P_0$ and its $k$ children $P_1 \dots P_k$. In the base model, $P_0$ can communicate with a single child at a time: it needs $c_i$ units of time to communicate a task to child $P_i$. Concurrently, $P_0$ can receives data from its own parent, say $P_{-1}$, requiring $c_{-1}$ time per task. We give three examples in Figures 3, 4 and 5: in the first, all children operate at full rate, and in the latter two, the communication bandwidth is the limiting factor.

**Proposition 1** *With the above notations, the minimal value of $w_{tree}$ for the fork graph is obtained as follows:*

1. *Sort the children by increasing communication times. Re-number them so that $c_1 \leq c_2 \dots \leq c_k$.*
2. *Let $p$ be the largest index so that $\sum_{i=1}^{p} \frac{c_i}{w_i} \leq 1$. If $p < k$ let $\epsilon = 1 - \sum_{i=1}^{p} \frac{c_i}{w_i}$, otherwise let $\epsilon = 0$.*
3. *Then $w_{tree} = \max\left(c_{-1}, \ \frac{1}{\frac{1}{w_0} + \sum_{i=1}^{p} \frac{1}{w_i} + \frac{\epsilon}{c_{p+1}}}\right)$*

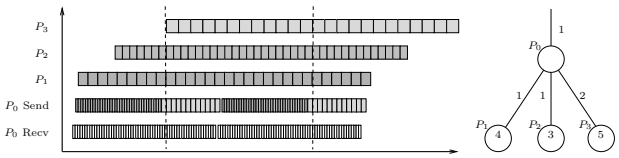


**Figure 3. First example without saturation of the communication bandwidth: all children can be kept fully active**

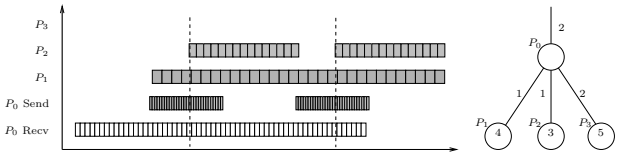


**Figure 4. Second example with saturation of the communication bandwidth: some children are partially idle due to the low bandwidth between $P_0$ and its parent**

Intuitively, the processors cannot consume more tasks than sent by $P_{-1}$, hence the first term of the maximum, i.e. $c_{-1}$. For the second term, when $p = k$ the result is expected: it basically says that children can be fed with tasks fast enough so that they are all kept computing steadily. However if $p < k$ the result is surprising: in the situation when the communication bandwidth is limited, some children will partially starve: these are those with slow communication rates, whatever their processing speeds. In other words, a slow processor with a fast communication link is to be preferred to a fast processor with a slow communication link.

**Proof** Suppose the optimal periodic schedule has a fixed, steady state pattern of period $T$: starting at time-step $t_0$, the whole pattern of computations and communications repeats every $T$ time-units, i.e. at time-step $t_0 + T$, $t_0 + 2T$, and so on. At the end of the proof, we will give upper bounds on $t_0$ and $T$ for (one particular) asymptotically optimal schedule.

During this period of $T$ units of time, let $x_{-1}$ denote the amount of time the node spends receiving tasks from its own parent $P_{-1}$, $x_0$ be the time spent computing, $x_1$ be the time sending tasks to child $P_1$, $x_2$ units of time sending tasks to child $P_2$, ..., and $x_k$ units of time sending tasks to child $P_k$. Because of periodicity and integral costs, all of the $x_i$'s are integers.

Because of the overlap hypothesis, the computation of $x_0$ tasks by $P_0$ doesn't diminish either the number

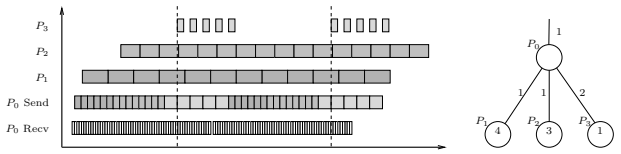


**Figure 5. Third example with saturation of the communication bandwidth: child $P_3$ is partially idle due to its high computation speed**

$x_{-1}$ it can receive from $P_{-1}$ nor $x_1, x_2, \ldots, x_k$. We have the following constraints for $x_0$: $0 \leq x_0 \leq T$, $x_0$ should be a multiple of $w_0$. The number of tasks that will be processed by $P_0$ is $\frac{x_0}{w_0}$.

We have similar constraints on $x_{-1}$: $0 \leq x_{-1} \leq T$, $x_{-1}$ should be a multiple of $c_{-1}$. The number of tasks that are sent to $P_0$ is $\frac{x_{-1}}{c_{-1}}$.

There are more constraints on $x_1, \ldots, x_k$. For $1 \leq i \leq k$, $0 \leq x_i \leq T$, and $x_i$ should be a multiple of $c_i$. The single-port hypothesis translates into $x_1 + x_2 + \ldots + x_k \leq T$. The number of tasks received by child $P_i$ is $\frac{x_i}{c_i}$; it can process these tasks within the time frame iff $\frac{x_i}{c_i} \times w_i \leq T$. Note that $P_i$ should not receive more tasks than it can process.

There is a final constraint that ensures the steady-state of the whole process: the number of tasks sent to the fork-join graph, i.e. $\frac{x_{-1}}{c_{-1}}$, should be equal to the number of tasks that it can consume, i.e. $\frac{x_0}{w_0} + \sum_{i=1}^{k} \frac{x_i}{c_i}$. This last quantity is exactly the quantity that we want to maximize.

Let us summarize the situation as a linear programming problem, where the objective function is the number of tasks consumed within the $T$ units of time:

$$\textbf{Maximize } \frac{x_0}{w_0} + \sum_{i=1}^{k} \frac{x_i}{c_i}$$
$$\textbf{subject to } \begin{cases} 0 \leq x_i \leq T \text{ for } -1 \leq i \leq k \\ \frac{x_i}{c_i} \times w_i \leq T \text{ for } 1 \leq i \leq k \\ \sum_{i=1}^{k} x_i \leq T \\ \frac{x_{-1}}{c_{-1}} = \frac{x_0}{w_0} + \sum_{i=1}^{k} \frac{x_i}{c_i} \end{cases}$$

Because everything is linear in $T$, we can normalize the problem and set $T = 1$: we now look for nonnegative *rational* values $x_{-1}, x_0, x_1, \ldots, x_k$ such that

$$\begin{array}{l} x_{-1} \leq 1 \\ x_0 \leq 1 \\ \sum_{i=1}^{k} x_i \leq 1 \\ x_i \leq \frac{c_i}{w_i} \text{ for } 1 \leq i \leq k \\ \frac{x_{-1}}{c_{-1}} = \frac{x_0}{w_0} + \sum_{i=1}^{k} \frac{x_i}{c_i} \end{array}$$

We can further normalize these equations by introducing the rates $R_i$: $R_{-1} = \frac{x_{-1}}{c_{-1}}$ is the rate of tasks per second received from the parent $P_{-1}$, $R_0 = \frac{x_0}{w_0}$ is the rate they are executed in the parent node $P_0$, and $R_i$, for $1 \leq i \leq k$, is the rate they are sent to and executed on the $i$-th child. We obtain the following formulation of the problem of determining the optimal task allocation for the base model:

**Base Problem: Maximize $\sum_{i=0}^{k} R_i$, subject to**
- **(B0)** $R_{-1} = \sum_{i=0}^{k} R_i$
- **(B1)** $R_{-1} c_{-1} \leq 1$
- **(B2)** $R_i \leq \frac{1}{w_i}$ for $0 \leq i \leq k$
- **(B3)** $\sum_{i=1}^{k} R_i c_i \leq 1$

Note that moving from the $x_i$'s to the rates $R_i$'s has a technical advantage: the new formulation nicely encompasses the case where some communication time $c_i$ is zero (which might be appropriate, for instance, for a shared-memory multiprocessor).

Let $R$ be the solution of the Base Problem. We claim that $R = \min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right)$ (unless $c_{-1} = 0$, in that case $R = \frac{1}{w_0} + S$) , where $S$ is the solution to the following problem:

**Auxiliary problem: Maximize $\sum_{i=1}^{k} R_i$, subject to**
- **(i)** $R_i w_i \leq 1$ for $1 \leq i \leq k$
- **(ii)** $\sum_{i=1}^{k} R_i c_i \leq 1$

Because the auxiliary problem is less constrained than the original one, we immediately have that $\frac{1}{w_0} + S \geq R$. We also have $\frac{1}{c_{-1}} \geq R$, because of (B0) and (B1), hence $\min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right) \geq R$. To show the reverse inequality, there are two cases, according to the value of $\min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right)$. Assume first that $\min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right) = \frac{1}{c_{-1}}$. Let $(R_1, \ldots, R_k)$ be the optimal solution of the auxiliary problem: $S = \sum_{i=1}^{k} R_i$ and $\frac{1}{w_0} + S \geq \frac{1}{c_{-1}}$. Let $\alpha = \frac{\frac{1}{c_{-1}}}{\frac{1}{w_0} + S} \leq 1$. Then $\left(\frac{1}{c_{-1}}, \frac{\alpha}{w_0}, \alpha R_1, \ldots, \alpha R_k\right)$ is a solution to the base problem whose objective function is equal to $\frac{1}{c_{-1}}$. Therefore $\frac{1}{c_{-1}} \leq R$.

Assume now that $\min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right) = \frac{1}{w_0} + S$. Let $(R_1, \ldots, R_k)$ be the optimal solution of the auxiliary problem: $S = \sum_{i=1}^{k} R_i$ and $\frac{1}{w_0} + S \leq \frac{1}{c_{-1}}$. Let $R_0 = \frac{1}{w_0}$ and $R_{-1} = \sum_{i=0}^{k} R_i$. Then $(R_{-1}, R_0, R_1, \ldots, R_k)$ is a solution to the base problem (note that (B1) is satisfied because of the hypothesis). Hence $\frac{1}{w_0} + S \leq R$. This concludes the reduction to the auxiliary problem.

We can now come to the solution of the auxiliary problem. As in the statement of the theorem, let $p$ be the largest index so that $\sum_{i=1}^{p} \frac{c_i}{w_i} \leq 1$. Let $R_i^* = \frac{1}{w_i}$ for $1 \leq i \leq p$. If $p < k$, let $R_{p+1}^* = \frac{\epsilon}{c_{p+1}}$, where $\epsilon = 1 - \sum_{i=1}^{p} \frac{c_i}{w_i}$. If $p+1 < k$, let $R_i^* = 0$ for $p+2 \leq i \leq k$. We claim that $(R_i^*)$ is the optimal solution of the optimization problem:

- First, it is indeed a solution. We have $R_{p+1}^* \leq \frac{1}{w_{p+1}}$ when $p < k$. This comes directly from the definition of $p$: since $\sum_{i=1}^{p+1} \frac{c_i}{w_i} > 1$, we have $\epsilon = 1 - \sum_{i=1}^{p} \frac{c_i}{w_i} < \frac{c_{p+1}}{w_{p+1}}$, hence $\frac{\epsilon}{c_{p+1}} \leq \frac{1}{w_{p+1}}$. As for (ii), $\sum_{i=1}^{j} R_i c_i = \sum_{i=1}^{j} \frac{c_i}{w_i} \leq 1$ for all $j \leq p$, by definition of $p$. And if $p < k$, $\sum_{i=1}^{p+1} R_i c_i = 1$, by definition of $R_{p+1}$.
- Second, it is the solution that maximizes the objective function. To see this, consider all the optimal solutions, i.e. all the solutions that achieve the optimal value of the objective function. Among these optimal solutions, consider one solution $(R_i^*)$ such that $R_1$ is maximal. Assume by contradiction that $R_1 < R_1^* = \frac{1}{w_1}$. Then there exists at least one index $j > 2$ such that $R_j > R_j^*$, otherwise the solution would not be optimal. Now, since the $c_i$ are sorted, we have $c_1 \leq c_j$. We do not change the value of the objective function if we let $R_1 = R_1 + \tau$ and $R_j = R_j - \tau$, where $\tau$ is an arbitrary small non-negative rational number. However, we do have a new solution to the optimization problem, because $(R_1 + \tau)c_1 + (R_j - \tau)c_j \leq R_1 c_1 + R_j c_j$. Hence we have an optimal solution with a larger $R_1$ than the original one, a contradiction. Hence we have shown that there exist optimal solutions such that $R_1 = R_1^*$. We restrict to such solutions without loss of generality and we iterate the process: we finally derive that $(R_i^*)$ is an optimal solution.

The optimal solution of the auxiliary problem is

$$S = \sum_{i=1}^{k} R_i = \sum_{i=1}^{p} \frac{1}{w_i} + \frac{\epsilon}{c_{p+1}}$$

The optimal solution of the base problem is

$$R = \min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right)$$

Finally, $w_{\text{tree}} = \frac{1}{R}$, which establishes our claim.

To conclude the proof, we only need to provide values for $t_0$ and $T$. Since we know the optimal solution $(R_i^*)$, we choose $T$ such that an integral number of tasks can be processed by each processor. We have $R_i^* = \frac{1}{w_i}$ for $0 \leq i \leq p$, hence $T$ must be a multiple of each $w_i$, $0 \leq i \leq p$. Since $R_{p+1}^* = \frac{\epsilon}{c_{p+1}}$, $T$ should be a multiple of $c_{p+1}$. Note that $T\epsilon$ is an integer if $T$ is a multiple of each $w_i$, $0 \leq i \leq p$. Finally, $T$ should be larger than $c_{-1}$. A valid value (though not necessarily the smallest possible value) for $T$ is

$$T = \text{lcm}(w_0, w_1, \ldots, w_p, c_{-1}, c_{p+1}).$$

It is not surprising that $c_1, \ldots, c_p$ do not appear in the expression for $T$, because the choice of $\epsilon$ ensures that $c_i \leq w_i$ for $1 \leq i \leq p$. Finally, we can construct a schedule that enters the steady-state behavior with $t_0 = T$: to see this, perform exactly the same pattern of communications but no computation at all from time-step 0 to $t_0 - 1 = T - 1$; each processor is then provided with exactly as many tasks as required to enter its steady-state.

A less formal (but much shorter) proof of Proposition 1 is the following: sorting the $c_i$ and feeding as many tasks as possible to the children taken in that order maximizes the number of tasks that are communicated to the children, hence the number of tasks that are processed by the children. Add those processed by the parent, and take the minimum with the input rate to derive the optimal value. ■

Note that the proof in Proposition 1 is fully constructive: the number of tasks to be computed by the parent and to be sent to each child is directly computed from the optimal solution $(R_i^*)$.

### 3.2 Arbitrary tree graphs

The best allocation of tasks to processor nodes is determined using a bottom-up traversal of the tree:

**Proposition 2** *Let $G = (V, E, w, c)$ be an arbitrary tree graph. The minimal value of $w_{tree}$ for the whole graph is obtained as follows:*

1. *Consider any sub-tree consisting of several leaves and their parent. Replace this tree with a single node whose weight is given by Proposition 1*
2. *Iterate the process until there remains a single node.*

*Then the minimal value is equal to the weight of the single node.*

**Proof** The proof is immediate: in steady state, a fork graph consisting of a parent and several leaves behaves exactly as a single node of weight determined by Proposition 1. For the root node, we can assume a link from a virtual parent with infinite capacity, i.e. $c_{-1} = 0$.

We can give (possibly unnecessarily large) bounds on the period and start-up times. The period $T$ of an arbitrary tree is at most the least common multiple of the periods of each fork graph encountered in the bottom-up traversal of the tree. And a bound on the start-up time is this period $T$ times the maximum depth of the tree. This time corresponds to a start-up strategy in which during the first $T$ time units, we send a period's worth of work to the depth one nodes (but don't execute any tasks), in the next $T$ time units we propagate this work to the depth two nodes and replenish the depth one nodes, and so on. ■

Again, the proof is fully constructive, and the optimal task allocation is computed using the bottom-up approach.

## 4 Conclusion and discussion

In this paper, we considered the problem of allocating a large number of independent, equal-sized tasks to a heterogeneous collection of computing resources. We assume the data for the tasks is initially located on a single resource. Such problems arise in collaborative computing efforts. This paper makes the following contributions to this problem:

- We suggest that it is natural to model the computing resources as a tree, whose nodes can have different speeds of computation and communication. Our model also allows nodes to have differing capabilities in terms of allowing overlap of computation and communication.
- For one particular model of overlap capabilities — the base model which allows computation and communication in two directions to overlap — we give an explicit solution to the problem of determining the allocation of tasks to nodes in the tree that maximizes the number of tasks executed per unit time in steady-state. This optimal solution is *bandwidth-centric*: tasks should be allocated to nodes in order of fastest communication time. This result may be counter-intuitive, it says the speed of the processors is irrelevant to choosing which processors to send tasks to.

In the extended version of the paper [1], we mention how other models of overlap capability can be reduced to the base model. The reductions work on a node-by-node basis, so it is simple to handle a heterogeneous system with nodes having not only differing speeds but also differing capabilities.

In [1] we also present simulation results of demand-driven task allocation heuristics. These results show that our bandwidth-centric method, where tasks are allocated only to nodes with sufficiently fast communication times, obtains better results than allocating tasks to all processors on a first-come, first serve basis. Even better results are given by giving a low priority to the processors that, in the optimal solution, are idle part of the time.

Our model does not directly address the dynamically changing nature of the grid, nor does it consider the total execution time from sending out the first task to receiving the result from the very last. Although both of these are important considerations, this paper nonetheless suggests a simple and powerful heuristic: allocation of tasks at each node should be bandwidth-centric. In other words, given limited bandwidth, tasks should be allocated locally at each node with priority given to *how fast they can be communicated* from the node to a child, and not based on the computational speed of the child.

## References


[1] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. Technical Report 2001-25, LIP, ENS Lyon, France, June 2001. Available at www.ens-lyon.fr/ ∼yrobert.

[2] F. Berman. High-performance schedulers. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 279–309. Morgan-Kaufmann, 1999.

[3] J. Cowie, B. Dodson, R.-M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, and J. Zayer. A world wide number field sieve factoring record: on to 512 bits. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology - Asiacrypt '96*, volume 1163 of *LNCS*, pages 382–394. Springer Verlag, 1996.

[4] Entropia. URL: `http://www.entropia.com`.

[5] D. G. Feitelson. High performance cluster computing. volume 1: Architecture and systems. In R. Buyya, editor, *The Grid: Blueprint for a New Computing Infrastructure*, pages 519–533. Prentice Hall PTR, 1999.

[6] B. Lowenkamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. In *Proceedings of the Seventh International Symposium on High Performance Distributed Computing*, 1998.

[7] Prime. URL: `http://www.mersenne.org`.

[8] SETI. URL: `http://setiathome.ssl.berkeley.edu`.

[9] R. Wolski, N. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(10):757–768, 1999.