

Bandwidth-Aware Resource Allocation for Computing Independent Tasks in Heterogeneous Computing Systems*

Bo Hong and Viktor K. Prasanna
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
{bohong, prasanna}@usc.edu

Abstract

In this paper, we consider the resource allocation problem for computing a large set of equal-sized independent tasks on heterogeneous computing systems. This problem represents the computation paradigm for a wide range of applications such as SETI@home and Monte Carlo simulations. Compute nodes in the system are heterogeneous and may communicate with each other at different speeds. We model such a system as a graph, which is capable of representing an arbitrary network topology. Our study focuses on the maximization of the steady state throughput of such systems. We show that, unlike the surprisingly difficult makespan minimization problem, the throughput maximization problem can be solved through a linear programming formulation. We also show that different operation scenarios of the compute nodes have intrinsic similarities. Our approach shows improved performance compared with the master/worker computation paradigm which assumes a tree-structured system topology.

KEY WORDS

Resource Allocation, Heterogeneous Computing System, Throughput, Linear Programming

1 Introduction

In this paper, we consider the problem of computing a large set of equal-sized independent tasks on a heterogeneous computing system. This problem represents the computation paradigm for a wide range of applications. Internet based distributed computing projects are among the most well-known examples of this computation paradigm. Examples of such research projects include SETI@home [8], Folding@home [9], data encryption/decryption [3], etc. This computation paradigm also applies to other more tightly coupled computations such as Monte Carlo simulations.

The computing system consists of a heterogeneous collection of compute nodes, connected via heterogeneous

network links. The network topology can be arbitrary. We model the system as an undirected graph, where each node in the graph represents a compute node and each edge in the graph represents a network link. A compute node needs to receive the source data for a task before executing this task. We assume that the source data for all tasks initially resides on a single node in the system, which we call the root node. A compute node can communicate with not only the root node (if such a network link exists), but also its neighbors. Every compute node thus needs to determine (1) where to get the tasks from and how many, (2) how many tasks to compute locally, and (3) where to transfer the rest of the tasks that it has received.

This is actually a resource allocation problem that can be reduced to the scheduling of a set of independent tasks on heterogeneous computing systems, for which various methods have been proposed to minimize the overall execution time (makespan) of all the tasks. As this makespan minimization problem is known to be NP-complete [7], designing effective heuristics and evaluating their performance become the key issues (e.g. [2, 10]).

Instead of minimizing the makespan, an alternate optimization objective is studied in this paper: maximization of the steady state system throughput. Maximization of steady state system throughput is not equivalent to the minimization of makespan, since the system may not operate at full speed during start up and trailing time. However, if the number of tasks is large, then the start up and the trailing time become negligible when compared with the overall computing time of all tasks. For applications that have a huge number of tasks such as SETI@home, system throughput, naturally, becomes the major concern.

There are some works that consider system throughput. Master/Worker paradigm is widely used to schedule independent tasks in heterogeneous computing systems and has been exploited by various research groups ([4, 11]) to maximize the throughput. In [1], a bandwidth-centric approach was proposed to maximize system throughput when the nodes are connected via a tree topology and all the tasks initially reside on the root node. Compared with these researches, the proposed study in this paper targets a more general problem that allows an arbitrary network

*Supported by the National Science Foundation under award No. ACI-0204046 and an equipment grant from Intel Corporation.

topology. Not only does this arbitrary network topology represent a more realistic heterogeneous computing system, it can also outperform the tree topology as it provides more choices to route the tasks through the system. (See Section 5 for the comparison of graph-structured systems and the tree-structured systems.) Further, since the tree-structured topology is a special case of an arbitrary topology, algorithms obtained from our study can be directly applied to those tree structured systems.

We show that, unlike the surprisingly difficult makespan minimization problem, the throughput maximization problem can be formulated as a linear programming problem, which can then be solved efficiently. Our study also shows that different operation scenarios of the compute nodes (see Section 4 for details of these scenarios) have intrinsic similarities. All the operation scenarios can be reduced to a base scenario where the computation and communication can be overlapped on the compute nodes, and the compute nodes can either send data to one neighbor or receive data from one neighbor.

Simulations have been conducted to verify the effectiveness of the proposed linear programming formulation method. The results show that by considering all the communication links in a system with an arbitrary network topology, we can improve the throughput by up to 2.92 times than if we restrict the communications to a tree-structured subset of the communication links. Based on the optimal solution for the throughput maximization problem, we have developed a simple bandwidth-aware heuristic to allocate the compute and communication resources. Simulations show that this heuristic increases the system throughput by up to 92% when compared with a first come first serve heuristic.

The rest of the paper is organized as follows. Section 2 describes our system model and formally states the optimization problem. In Section 3, we discuss our linear programming formulation that maximizes the system throughput. In Section 4, we show how to reduce other operation scenarios of the compute nodes to the base scenario discussed in Section 3. Experimental results are shown in Section 5. Concluding remarks are made in Section 6.

2 System Model and Problem Statement

The system is represented by an undirected graph $G(V, E)$. Each node $V_i \in V$ in the graph represents a compute node. The weight of V_i , denoted by w_i , represents the processing power of node V_i , i.e. V_i can perform one unit of computation in $1/w_i$ time. Each edge $E_{ij} \in E$ in the graph represents a bi-directional network link between V_i and V_j . $E_{ij} = E_{ji}$. The weight of E_{ij} , denoted by l_{ij} , represents the communication bandwidth of link E_{ij} , i.e. link E_{ij} can transfer one unit of data from V_i to V_j (or from V_j to V_i) in $1/l_{ij}$ time. We use A_i to denote the adjacent nodes of V_i in G , i.e. $A_i = \{V_k | \exists E_{ik} \in E\}$.

The compute nodes may work in different scenarios. First, the computation and the communication may or may

not be overlapped on the compute nodes. There are also two possible operation scenarios for the network interface of a compute node: it can be full-duplex or half-duplex. A compute node can send and receive data concurrently if it has a full-duplex network interface. A compute node with a half-duplex network interface can send and receive data, but the send and receive operations cannot be performed at the same time. When multiple network links connect to a single node, this node may receive(send) data through these links either concurrently or sequentially, i.e. the network interface of this node may or may not support concurrent incoming(outgoing) communications with multiple neighbors.

The possible scenarios will be discussed in this paper. For the discussion in this section, we assume for the time being that all nodes/links operate as follows: (1) computation and communication can be fully overlapped on the compute nodes, (2) network interface of the compute nodes are half-duplex, (3) at any time instance, a compute node can only send data to one of its neighboring nodes and receive data from one of its neighboring nodes. We denote this scenario as the *base scenario*. Other scenarios will be discussed in Section 4.

Without loss of generality, we assume each task has one unit of source data and requires one unit of computation. (We can normalize the values of w_i and l_{ij} if the tasks are not unit-sized.) So a task is transferred over a network link means one unit of data is transferred. A task is computed by a compute node means one unit of computation is performed. A compute node can compute a task only after receiving the source data. Initially, node V_0 holds the source data for all the tasks. V_0 is called the root node. Each node V_i in the system receives tasks from a subset of its neighbors (V_0 could be the neighbor of some nodes), computes a subset of the tasks it received, and sends the remaining tasks to (possibly) another subset of its neighbors. V_0 is different from other nodes since it only sends out tasks and does not receive any tasks from any of its neighbors.

The steady-state throughput of the system is defined as the maximum number of tasks that can be processed by the system in one unit of time. We are now interested in the following problem: given a time interval T , what is the maximum number of tasks that can be processed by the system G ? Let $f(V_i, V_j)$ denote the number of tasks to be transferred from V_i to V_j during this time interval. To simplify our discussion, if the actual data transfer is from V_i to V_j , we define $f(V_j, V_i) = -f(V_i, V_j)$. We have the following constraints:

1. $|f(V_i, V_j)/l_{ij}| \leq T$ for \forall edge E_{ij} . This is because E_{ij} can transfer at most l_{ij} unit of data in one unit of time.
2. $\sum_{V_k \in A_i} f(V_k, V_i) \geq 0$ for $V_i \in V - \{V_0\}$. This condition says that no intermediate nodes can generate tasks. This constraint does not apply to V_0 .
3. $\sum_{V_k \in A_i} |f(V_i, V_k)/l_{ik}| \leq 1$ for $V_i \in V$. This is because V_i has a half-duplex network interface and all

the send and receive operations have to be performed serially.

4. $\sum_{V_k \in A_i} f(V_k, V_i)/w_i \leq T$ for $V_i \in V - \{V_0\}$. We can see that $\sum_{V_k \in A_i} f(V_k, V_i)$ is the total number of tasks that V_i has kept locally (tasks received minus tasks sent out). This condition says that any node V_i should not keep more tasks than it can compute, otherwise the number of un-computed tasks on this node will increase monotonically as time advances.

The total number of tasks computed by the system is $T \times w_0 + \sum_{V_i \in V - \{V_0\}} (\sum_{V_k \in A_i} f(V_k, V_i))$. Since we are interested in the throughput of the system, we can normalize time interval T to 1 and obtain the formal problem statement as follows:

Base Problem: Given an undirected graph $G(V, E)$ where V is the set of nodes and E is the set of edges. Node V_i has weight w_i . Edge E_{ij} connects V_i and V_j , and has weight l_{ij} . $w_i > 0$. $l_{ij} > 0$. If there is no edge between node V_k and V_m , we define $l_{km} = 0$. We are to find a real-valued function $f : V \times V \rightarrow R$ that satisfies:

1. $f(V_j, V_i) = -f(V_i, V_j)$ for $\forall V_i, V_j \in V$
2. $f(V_i, V_j) \leq l_{ij}$ for $\forall V_i, V_j \in V$
3. $\sum_{V_k \in A_i} |f(V_i, V_k)/l_{ik}| \leq 1$ for $\forall V_i \in V$
4. $0 \leq \sum_{V_k \in A_i} f(V_k, V_i) \leq w_i$ for $\forall V_i \in V - \{V_0\}$

and maximizes

$$\mathcal{W} = w_0 + \sum_{V_i \in V - \{V_0\}} \left(\sum_{V_k \in A_i} f(V_k, V_i) \right) \quad (1)$$

Since $l_{ij} = 0$ when edge E_{ij} does not exist, conditions 1 and 2 imply that $f(V_i, V_j) = f(V_j, V_i) = 0$ when there is no edge between V_i and V_j .

3 A Linear Programming Formulation

In this section, we show that the Base Problem (when all the compute nodes work in the base scenario) can be reduced to a linear programming problem. This is done by converting graph $G(V, E)$ in the Base Problem using Procedure 1. We denote the resulting graph of this conversion as $G'(V', E')$.

Procedure 1: For each node $V_i \in V$, create a corresponding node $V'_i \in V'$. V'_i has weight w_i . For each undirected edge $E_{ij} \in E$, create two directed edges E'_{ij} and E'_{ji} in E' , where E'_{ij} is from V'_i to V'_j and E'_{ji} is from V'_j to V'_i . Both E'_{ij} and E'_{ji} have weight l_{ij} .

With this transformation, we have the following linear programming problem:

Problem 1: Given a directed Graph $G(V, E)$. The weight of node V_i is w_i . Edge E_{ij} has weight l_{ij} . $w_i \geq 0$. $l_{ij} > 0$. Find a real-valued function $f : E \rightarrow R$ that satisfies:

1. $0 \leq f(E_{ij}) \leq l_{ij}$ for $\forall E_{ij} \in E$
2. $\sum_k f(E_{ik})/l_{ik} + \sum_k f(E_{ki})/l_{ki} \leq 1$ for $\forall V_i \in V$
3. $0 \leq \sum_k f(E_{ki}) - \sum_k f(E_{ik}) \leq w_i$ for $\forall V_i \in V - \{V_0\}$

and maximizes

$$\mathcal{W} = w_0 + \sum_k f(E_{0k}) - \sum_k f(E_{k0}) \quad (2)$$

The following proposition shows that the Base Problem and Problem 1 are equivalent. $\mathcal{W}_B(G, V_0)$ represents the maximum throughput for the base problem, given G as the input graph and V_0 as the root node. $\mathcal{W}_1(G, V_0)$ represents the maximum throughput for Problem 1, given G as the input graph and V_0 as the root node.

Proposition 3.1 *If $G(V, E)$ is transformed to $G'(V', E')$ using Procedure 1, then*

$$\mathcal{W}_B(G, V_0) = \mathcal{W}_1(G', V'_0)$$

Proof: We use the notation in Procedure 1 to denote the nodes/edges in G and their corresponding nodes/edges in G' .

Suppose $f : V \times V \rightarrow R$ is a feasible solution for the Base Problem instance (G, V_0) . We map it to a feasible solution $f' : E' \rightarrow R$ for Problem 1 instance (G', V'_0) as follows: if $f(V_i, V_j) \geq 0$, then we set

$$f'(E'_{ij}) = f(V_i, V_j), \quad f'(E'_{ji}) = 0$$

It is easy to verify that such an f' is a feasible solution for Problem 1 instance (G', V'_0) and that f' leads to the same throughput as f . (We only need to check condition 1, 2, and 3 of Problem 1. This checking process is straight forward and hence omitted here.)

Suppose $f' : E' \rightarrow R$ is a feasible solution for an instance of Problem 1 (G', V'_0) . We map it to a feasible solution $f : V \times V \rightarrow R$ for Base Problem instance (G, V_0) using the following equation:

$$f(V_i, V_j) = f'(E'_{ij}) - f'(E'_{ji})$$

It is also easy to verify that such an f is a feasible solution for Base Problem instance (G, V_0) and that it has the same throughput as f' . ■

Problem 1 is an instance of the well-studied linear programming problem. Various algorithms have been proposed to solve the linear programming problems. These include the Simplex algorithm that is easy to implement and has excellent average case performance, and the interior point algorithms that guarantee a polynomial time complexity. This shows that the throughput maximization problem can be solved very efficiently when the compute nodes work in the base scenario. In the next section, we will show that other operation scenarios of the compute nodes can be reduced to the base scenario.

4 Other Operation Scenarios of the Compute Nodes

Based on the discussion in Section 2, the operation scenario of a compute node is determined by the following four criteria: (1) whether computation and communication can be overlapped, (2) whether the network interface is full-duplex or half-duplex, (3) whether the network interface can receive data from multiple neighbors concurrently, and (4) whether the network interface can send data to multiple neighbors concurrently. We show that our linear programming formulation captures all the operation scenarios of the compute nodes.

Possible combinations of the above four criteria give us 16 operation scenarios. However, many of these scenarios do not have practical importance. For example, if a compute node can receive data from multiple neighbors concurrently, it is very unlikely that this node can send data to only one neighbor at a time. When the network interface of a compute node works in half-duplex mode, it is possibly a legacy device and may not be able to handle multiple connections concurrently. Therefore, in the discussion below, we will focus on the following four scenarios.

Scenario 1: Overlap-able computation and communication, half-duplex network interface, serial send, and serial receive. This is the base scenario and has already been discussed.

Scenario 2: Overlap-able computation and communication, full-duplex network interface, serial send, and serial receive. A compute node working in this scenario can send and receive data concurrently, although it cannot send to multiple neighbors concurrently or receive from multiple neighbors concurrently. A problem formulation for this scenario can be obtained (the method is similar to that for the base scenario. Details are omitted here due to space limitations) as follows:

Problem 2: Given a directed Graph $G(V, E)$. The weight of node V_i is w_i . Edge E_{ij} has weight l_{ij} . $w_i \geq 0$. $l_{ij} > 0$. Find a real-valued function $f_2 : E \rightarrow R$ that satisfies:

1. $0 \leq f_2(E_{ij}) \leq l_{ij}$ for $\forall E_{ij} \in E$
2. $\sum_k f_2(E_{ik})/l_{ik} \leq 1$ for $\forall V_i \in V$
3. $\sum_k f_2(E_{ki})/l_{ki} \leq 1$ for $\forall V_i \in V$
4. $0 \leq \sum_k f_2(E_{ki}) - \sum_k f_2(E_{ik}) \leq w_i$ for $\forall V_i \in V - \{V_0\}$

and maximizes

$$\mathcal{W} = w_0 + \sum_k f_2(E_{0k}) - \sum_k f_2(E_{k0}) \quad (3)$$

Problem 2 can be reduced to Problem 1. This is done through the following transformation: for every node V_i in Problem 2, split it into two nodes V_i^1 and V_i^2 . V_i^1 has weight 0 and V_i^2 has weight w_i . For every edge E_{ij} in Problem 2, replace it with another edge E'_{ij} that goes from V_i^2 to V_j^1 . E'_{ij} has weight l_{ij} . Finally, add a new edge

(denoted as E'_i) with weight ∞ from V_i^1 to V_i^2 . This transformation is illustrated in Figure 1.

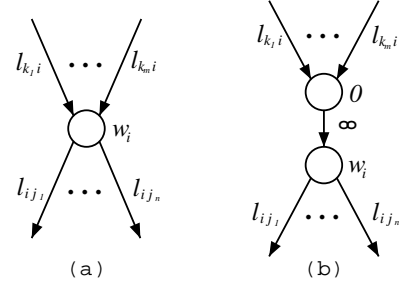


Figure 1. Illustration of reducing Problem 2 to Problem 1. (a) Node V_i in Problem 2. (b) Equivalent mapping of V_i to create an instance of Problem 1.

We use f_1 to denote the function that we search for Problem 1. Suppose the transformed graph is the input to an instance of Problem 1, then the constraints related to node V_i^1 and V_i^2 are as follows:

1. $0 \leq f_1(E'_{ik}) \leq l_{ik}$
2. $0 \leq f_1(E'_{ki}) \leq l_{ki}$
3. $0 \leq f_1(E'_i) \leq \infty$
4. $\sum_k f_1(E'_{ik})/l_{ik} + f_1(E'_i)/\infty \leq 1$
5. $f_1(E'_i)/\infty + \sum_k f_1(E'_{ki})/l_{ki} \leq 1$
6. $0 \leq \sum_k f_1(E'_{ki}) - f_1(E'_i) \leq 0$
7. $0 \leq f_1(E'_i) - \sum_k f_1(E'_{ik}) \leq w_i$

Constraint 3 above can be ignored since it is implied by constraint 1 and 6. In Constraint 4 and 5, ∞ appears in the denominator solely for the purpose of explanation. Because the newly added links have weight ∞ , terms $f(E'_i)/\infty$ and $f(E'_i)/\infty$ are 0. These constraints translate to:

1. $0 \leq f_1(E'_{ik}) \leq l_{ik}$
2. $0 \leq f_1(E'_{ki}) \leq l_{ki}$
3. $\sum_k f_1(E'_{ik})/l_{ik} \leq 1$
4. $\sum_k f_1(E'_{ki})/l_{ki} \leq 1$
5. $0 \leq \sum_k f_1(E'_{ki}) - \sum_k f_1(E'_{ik}) \leq w_i$

which are nothing but the constraints for Problem 2 if we let $f_2(E_{ik}) = f_1(E'_{ik})$ and $f_2(E_{ki}) = f_1(E'_{ki})$. This shows that Problem 2 is a special case of Problem 1.

Scenario 3: Overlap-able computation and communication, full-duplex network interface, concurrent multiple send, and concurrent multiple receive.

We need to further refine our model when a compute node can communicate with multiple neighboring nodes concurrently. Suppose a compute node connects to 5 other nodes, each through a 100 Mb/s link. It would be unrealistic to assume that this node can send or receive data at 500 Mb/s. A more reasonable model is that this node can communicate to only one of its neighbors at 100 Mb/s, or to all

five neighbors concurrently, but at 20 Mb/s each. Therefore, if a compute node V_i can communicate with multiple neighboring nodes concurrently, we introduce another two parameters c_i^{in} and c_i^{out} . These two parameters indicate the capability of V_i 's network interface to send and receive data: within one unit of time, at most c_i^{in} (c_i^{out}) units of data can flow into (out of) V_i .

Similar to Scenario 2, the following problem formulation can be obtained for this scenario:

Problem 3: Given a directed Graph $G(V, E)$. The weight of node V_i is w_i . Edge E_{ij} has weight l_{ij} . $w_i \geq 0$. $l_{ij} > 0$. Find a real-valued function $f_3 : E \rightarrow R$ that satisfies:

1. $0 \leq f_3(E_{ij}) \leq l_{ij}$ for $\forall E_{ij} \in E$
2. $\sum_k f_3(E_{ik}) \leq c_i^{out}$ for $\forall V_i \in V$
3. $\sum_k f_3(E_{ki}) \leq c_i^{in}$ for $\forall V_i \in V$
4. $0 \leq \sum_k f_3(E_{ki}) - \sum_k f_3(E_{ik}) \leq w_i$ for $\forall V_i \in V - \{V_0\}$

and maximizes

$$\mathcal{W} = w_0 + \sum_k f_3(E_{0k}) - \sum_k f_3(E_{k0}) \quad (4)$$

Problem 3 can be reduced to Problem 2 by using the following transformation: for every node V_i in Problem 3, split it into three nodes V_i^0 , V_i^1 , and V_i^2 . Both V_i^0 and V_i^2 have weight 0 and V_i^1 has weight w_i . Add edge ($V_i^0 \rightarrow V_i^1$) with weight c_i^{in} . Add edge ($V_i^1 \rightarrow V_i^2$) with weight c_i^{out} . For every edge E_{ij} in Problem 3, replace it with the following combined component: edge E_{ij}^1 (weight ∞) \rightarrow node V_{ij}^1 (weight 0) \rightarrow edge E_{ij}^2 (weight l_{ij}) \rightarrow node V_{ij}^2 (weight 0) \rightarrow edge E_{ij}^3 (weight ∞). In this combined component, double subscripts are used to denote the newly inserted nodes V_{ij}^1 and V_{ij}^2 . This transformation is illustrated in Figure 2. Without introducing any confusion, names of the nodes and edges are omitted to make Figure 2 more legible.

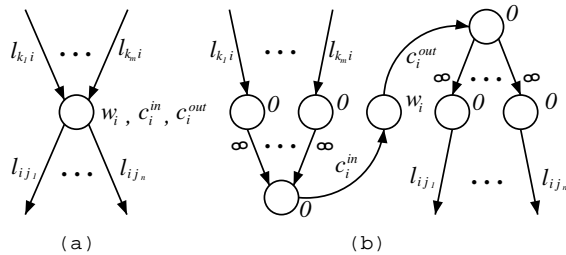


Figure 2. Illustration of reducing Problem 3 to Problem 2. (a) Node V_i in Problem 3. (b) Equivalent mapping of V_i to create an instance of Problem 2.

Similar to scenario 2, we can show that such a transformation reduces Problem 3 to a special case of problem 2. Details are omitted here due to space limitations.

Scenario 4: Non-overlap-able computation and communication, half-duplex network interface, serial send, and serial receive.

Similarly, we have the following problem formulation for this scenario:

Problem 4: Given a directed Graph $G(V, E)$. The weight of node V_i is w_i . Edge E_{ij} has weight l_{ij} . $w_i \geq 0$. $l_{ij} > 0$. Find a real-valued function $f_4 : E \rightarrow R$ that satisfies:

1. $0 \leq f_4(E_{ij}) \leq l_{ij}$ for $\forall E_{ij} \in E$
2. $\sum_k f_4(E_{ik})/l_{ik} + \sum_k f_4(E_{ki})/l_{ki} + (\sum_k f_4(E_{ki}) - \sum_k f_4(E_{ik}))/w_i \leq 1$ for $\forall V_i \in V$
3. $\sum_k f_4(E_{ki}) - \sum_k f_4(E_{ik}) \geq 0$ for $\forall V_i \in V - \{V_0\}$

and maximizes

$$\mathcal{W} = w_0 + \sum_k f_4(E_{0k}) - \sum_k f_4(E_{k0}) \quad (5)$$

Problem 4 can be reduced to Problem 1. This is done through the following transformation: for every node V_i in Problem 4, add another node V_i' with weight w_i and change the weight of V_i to 0. Add an edge ($V_i \rightarrow V_i'$) with weight w_i . This transformation is illustrated in Figure 1.

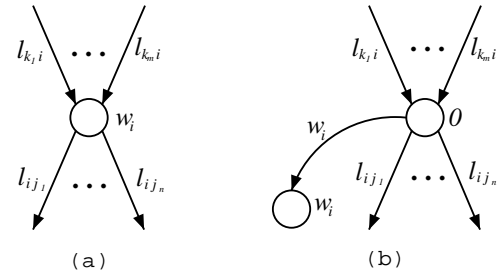


Figure 3. Illustration of reducing Problem 4 to Problem 1. (a) Node V_i in Problem 4. (b) Equivalent mapping of V_i to create an instance of Problem 1.

It can be proved that such a transformation reduces Problem 4 to a special case of Problem 1.

Because our linear programming based approach takes into account the constraints on both the computation and the communication capabilities of the resources, it not only reflects the compute capability of the compute nodes, but is also bandwidth-aware. We will show in the next section how this bandwidth-awareness is important to the system performance (in our case, the system throughput).

5 Experimental Results

As we have discussed in Section 4, different operation scenarios of the compute nodes can be reduced to the base scenario. Therefore, we only simulated the base scenario and the simulation results can be generalized to other scenarios. The linear programming formulation discussed in Section 3 are used in the simulations to find the optimal resource allocations.

As can be expected, we first show that graph-structured systems can achieve a higher throughput than tree-structured systems. We simulated two cases. In case

Table 1. Comparison of graph-structured and tree-structured systems. A graph is first generated and the tree is obtained by performing breadth first search.

$w_{max} = 0.1$					
link density	0.04	0.08	0.12	0.16	0.20
$n = 20$	1.16	2.96	2.60	2.35	3.65
$n = 40$	3.65	3.18	1.89	2.30	1.71
$n = 60$	2.09	1.72	1.49	1.74	1.55
$n = 80$	2.19	1.46	1.59	1.46	1.44
$w_{max} = 0.2$					
link density	0.04	0.08	0.12	0.16	0.20
$n = 20$	1.04	1.07	2.05	2.20	1.52
$n = 40$	1.86	1.25	1.71	1.24	1.55
$n = 60$	2.10	1.54	1.61	1.37	1.37
$n = 80$	1.37	1.21	1.37	1.22	1.22

1, we first randomly generate a graph-structured system. A graph is represented by its adjacency matrix A where each non-zero entry a_{ij} represents the bandwidth of the corresponding link l_{ij} . If $a_{ij} = 0$, then link (V_i, V_j) does not exist. The graph is generated as follows: Initially, all entries in A are set to 0. Then a set of entries is randomly selected. Each selected entry is assigned a value that is uniformly distributed between 0 and 1. We compare the performance of the graph-structured system against one of its spanning trees. Search for a spanning tree that has the highest system throughput among all the spanning trees is critical to the system performance. However, to the best of our knowledge, we are not aware of any efficient algorithms to solve this problem. In our experiments, we use a breadth first search (BFS) tree. Intuitively, a BFS tree attempts to find the shortest path for all the nodes to communicate with the root node. In case 2, a tree-structured system is first constructed. For every non-leaf node in the tree, its number of children is randomly determined between 1 and 5. We compare the performance of such a tree-structured system against a graph-structured system that is constructed by randomly adding links to the nodes in the tree. We limit the number of children for the nodes so that the tree can have multiple levels. In both Case 1 and 2, l_{ij} , c_i^{in} , c_i^{out} are uniformly distributed between 0 and w_{max} , where $1/w_{max}$ represents the average computation/communication ratio of the tasks.

The simulation results are shown in Table 1 and 2, where n is the number of nodes in the system. Data in the tables shows the ratio between the throughput of graph-structured systems and the throughput of the corresponding tree-structured system. *Link density* in Table 1 and 2 represents the number of edges in the system and is normalized as $\frac{|E|}{|V| \cdot (|V|-1)/2}$. Each reported data is the average over 100 randomly generated systems. As can be seen from these results, utilizing communication links in a general graph-structured system can improve the system throughput as

Table 2. Comparison of graph-structured and tree-structured systems. A tree is first generated and the graph is obtained by randomly adding links to the nodes in the tree.

$w_{max} = 0.1$					
link density	0.04	0.08	0.12	0.16	0.20
$n = 20$	1.26	1.50	1.49	1.70	1.78
$n = 40$	2.08	1.82	1.47	2.29	1.65
$n = 60$	1.46	1.59	1.92	2.09	1.44
$n = 80$	1.57	1.73	1.92	1.56	2.14
$w_{max} = 0.2$					
link density	0.04	0.08	0.12	0.16	0.20
$n = 20$	1.41	1.32	1.65	1.49	1.57
$n = 40$	1.60	1.56	1.71	1.28	3.92
$n = 60$	1.26	1.61	3.41	1.55	2.04
$n = 80$	1.63	1.62	1.60	1.57	2.58

much as 2.92 times if we are not limited to a tree topology.

The second set of simulations show that considering task transferring cost when allocating the resources leads to higher system throughput. More importantly, we show that the proposed resource allocation can be implemented easily.

By using the information provided by our linear programming formulation, we design a bandwidth-aware heuristic. The performance of this bandwidth-aware heuristic is compared with that of the first come first serve (FCFS) heuristic, which is a widely used heuristic to schedule independent tasks to heterogeneous computing systems. These two heuristic share the same outline: each node in the system keeps a task buffer. At the beginning of the computation, only the task buffer at the root node is non-empty. It contains all the tasks to be computed by the system. At any time instance, for each node in the system, (1) if the task buffer is not empty and the compute node is free, then remove one task from the task buffer and compute the task; (2) if the task buffer is still not empty, check the neighbors. If any neighbor has an empty task buffer, make a decision on whether to send a task to that neighbor or not, and transfer one task if the decision is 'yes'.

The two heuristics differ in step 2 in the above outline. In the first come first serve heuristic, any neighbor with an empty buffer is a candidate to transfer a task. A compute node is assigned a task whenever it becomes free. Hence the more powerful a compute node is, the more tasks it may get. This heuristic only considers the compute power of the nodes and ignores the communication bandwidth among the nodes. In our bandwidth-aware heuristic, a throughput maximization problem is first solved by using the linear programming formulation. Neighbor V_j of node V_i can receive tasks from V_i only if the solution of the throughput maximization problem shows a positive data flow from V_i to V_j , i.e. when $f(E_{ij}) > 0$. This heuristic only se-

lects those nodes and links that can maximize the system throughput. This selection is determined through the linear programming formulation.

The graphs were generated as in Case 1 of the first set of experiments. Initially, there are 1000 tasks on the root node. Figure 4 compares the throughput of the two heuristics. The reported throughput is calculated as the number of tasks computed in one unit of time ($1000/\text{overall_execution_time}$). Each data point in Figure 4 is an average over 100 randomly generated graphs. As can be seen from this figure, our bandwidth-aware heuristic increases the system throughput by up to 92% compared with the FCFS heuristic.

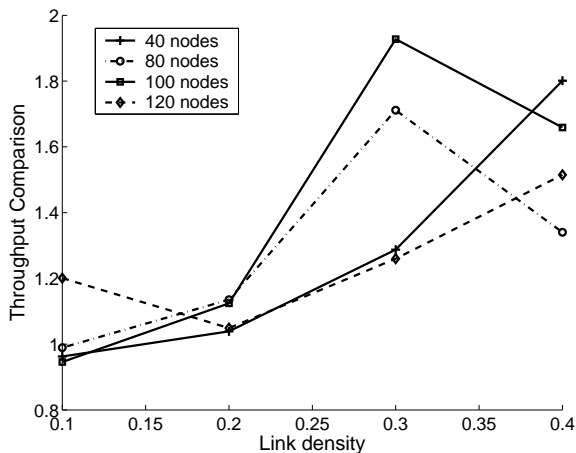


Figure 4. Comparison between our bandwidth-aware heuristic and the first come first serve (FCFS) heuristic. Data points in this figure show the ratio between the throughput of the bandwidth-aware heuristic and that of the FCFS heuristic.

6 Conclusion and Discussions

In this paper, we studied the problem of computing a large set of equal-sized independent tasks on a heterogeneous computing system. We studied systems with arbitrary network topology and modeled the systems as undirected graphs. Instead of minimizing the overall execution time of all tasks, we focus on the maximization of the steady state system throughput. We showed that the throughput maximization problem can be solved efficiently through a linear programming formulation. Our study also reveals that different operation scenarios of the compute nodes have intrinsic similarities and can be reduced to the base scenario. The effectiveness of the proposed bandwidth-aware resource allocation approach were verified through simulations.

In [5], we presented our results for the throughput maximization problem when the compute nodes work in Scenario 3 discussed in Section 4. We show that in this scenario, the linear programming formulation discussed in Section 3 can be further reduced to a network flow maxi-

mization problem, for which we can use various efficient algorithms. A more interesting result for this scenario is that we can develop an autonomous and distributed algorithm to maximize the system throughput [6].

Future studies need to consider the impact of communication/computation ratios of the tasks. This is because in real life applications, the size of the tasks is not given, but needs to be determined to maximize the system throughput - and the size of the tasks translates to the communication/computation ratios of the tasks. We also need to consider the scenario where the initial source data for the tasks can reside on multiple nodes. We can easily extend our linear programming formulation by adding a pseudo root node. However, for this situation, we need to find out which nodes to use as the initial data pool and how much data to place on each of these nodes.

References

- [1] O. Beaumont, A. Legrand, Y. Robert, L. Carter, and J. Ferrante. Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms. *International Parallel and Distributed Processing Symposium (IPDPS)*, April 2002.
- [2] T. D. Braun, H. J. Siegel, and N. Beck. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.
- [3] Distributed.net. <http://www.distributed.net>.
- [4] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Evaluation of an Adaptive Scheduling Strategy for Master-Worker Applications on Clusters of Workstations. *7th International Conference on High Performance Computing (HiPC 2000)*, December 2000.
- [5] B. Hong and V. K. Prasanna. Bandwidth-Aware Resource Allocation for Heterogeneous Computing Systems to Maximize Throughput. *International Conference on Parallel Processing (ICPP 2003)*, 2003.
- [6] B. Hong and V. K. Prasanna. Autonomous Resource Allocation for Heterogeneous Systems to Compute Independent Tasks. *submitted to International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.
- [7] O. Ibarra and C. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM*, 24(2):280–289, 1977.
- [8] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. SETI@home-Massively Distributed Computing for SETI. *Computing in Science and Engineering*, January 2001.
- [9] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande. *Folding@Home and Genome@Home: Using Distributed Computing to Tackle Previously Intractable Problems in Computational Biology, Computational Genomics*, Richard Grant, editor. Horizon Press, 2002.
- [10] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.
- [11] G. Shao, F. Berman, and R. Wolski. Master/Slave Computing on the Grid. *9th Heterogeneous Computing Workshop*, May 2000.