Computing Real Time Jobs in P2P Networks

Jingnan Yao, Jian Zhou, Laxmi Bhuyan
Department of Computer Science and Engineering
University of California, Riverside
{jyao, jianz, bhuyan}@cs.ucr.edu

Abstract— **In this paper, we present a distributed computing framework designed to support higher quality of service and fault tolerance for processing deadline-driven tasks in a P2P environment. Our proposed strategy strives to build an open infrastructure that is accessible by ordinary users for both cycle donation and consumption. For jobs that fail to be locally accommodated, the proposed scheduler MET (Maximum Efficiency Tree) builds a dynamic multi-level resource tree with minimal yet sufficient power to process the job prior to its deadline. The peer selection policy is based on a joint evaluation of the computational power and communication bandwidth at the nodes. Further, with an optimal load sharing scheme, the resulting resource tree is guaranteed to be power efficient.**

The proposed computing protocol offers an approach for utilizing idle computing cycles of peer computers on the Internet in a P2P manner. The protocol exhibits three attractive features - decentralized operation, optimized load balancing and guaranteed resource utilization. Extensive simulation experiments are conducted to study the effectiveness of the proposed framework under various network conditions. We compare our strategy with two other tree construction algorithms, namely MST (Minimum Spanning Tree) and MCT (Maximum Computation Tree). It is demonstrated that MET outperforms both MST and MCT consistently. Further, sensitivity results with random node failure/join are also furnished.

I. Introduction

With the advancement in computer technology, the computational power, storage space and communication bandwidth available on desktop machines have been growing dramatically. Millions of such resource-rich machines remain idle without being fully utilized locally. Peer-to-Peer (P2P) networks [1], [2], [3], among many other distributed computing models, exhibit good scalability and flexibility. They have been proven to be an efficient and successful way for file sharing over the Internet.

P2P computing [13], [16], [15] offers the opportunity to aggregate the unused computer cycles that are scattered on the network to support distributed computing. P2P characteristics like decentralized control, self-autonomy and load balancing make it very attractive for large-scale distributed applications. Hence, P2P computing offers an exciting new challenge for P2P networks beyond traditional information sharing and content distribution applications. As such, various applications such as portfolio pricing, market and credit evaluation, media transcoding [13], [12], [20] and many other computation intensive grid applications are being deployed over the Internet. Peer to Peer networks and grids are both distributed computing models that enable pooling and coordinated usage of large sets of distributed resources. In contrast to Grid computing [9], P2P computing offers better scalability and flexibility while compromising in reliability and quality of service, due to the autonomy and dynamics of its participants. A number of Internet based distributed computing projects, such as SETI@Home [1], Avaki [4] and Condor [5], have demonstrated the feasibility of this approach.

Most of the research work in the area of P2P have focussed on developing communication protocols and platforms [2], [3], [6] to accomplish data sharing and exchange among peers. There are few mechanisms for explicitly gathering and allocating remote cycles for computing purposes. Most of the cycle sharing systems demand a centralized web site and are limited to members of participating institutions [1], [5]. Their scheduling schemes are adhoc without a clear analysis of the requirements and prediction of the total execution time. They also lack a systematic procedure for recruiting resources from the large pool of Internet nodes. Therefore, decentralized job submission models and job scheduling strategies have become very attractive research topics [7], [15] to allow CPU cycle sharing among multiple jobs originating from ordinary users. Moreover, these applications often require predictable performance because tasks in these applications have deadlines to be met [12]. These real-time issues have not been addressed earlier. Hence, supporting them in a P2P environment with unpredictable latencies and varying resource availability presents a number of challenges in managing the networking resources and scheduling the task executions across the network. To effectively utilize the idle resources, efficient job partitioning, resource identification and load balancing algorithms have to be developed while considering the communication overhead to establish such a P2P distributed computing system.

Tremendous amount of job scheduling work has been done in the field of parallel processing. Given a generic multiprocessor topology and configuration, their goal is to come up with the optimal scheduling that minimizes the total processing time. On the other hand, researchers in the area of grid and P2P computing have focussed on identifying good peers [14], [11] based on the capability and reliability of their neighbors for possible resource sharing. As a result, the design of load sharing algorithms for large scale distributed systems is often isolated from the resource location process. Due to the absence of a centralized scheduler in such networks, it is also hard to develop load sharing algorithms that can optimize the performance as the nodes are usually connected in an arbitrary fashion. We are thus motivated to design an effective load sharing mechanism for P2P networks that identifies the most

[1]This research has been supported by NSF grant CNS 0509440.

efficient resource pool with an optimized load scheduling. Effectively, as the selection of peers is targeted for an efficient load sharing purpose, we focus to explore the maximum network utilization by building a resource tree with maximum efficiency. Starting from where the jobs originate, we attempt to build a dynamic multi-level resource tree on which the jobs can be properly shared. There are two specific questions that need to be answered while building this tree, namely,

- How to build the tree of peers taking into account their computational availability and communication bandwidth.
- When do we stop recruiting peers in the resource tree based on the real time requirement of the job.

In this paper, selection of peers is based on a combined evaluation of the available computational power and communication bandwidth. We show that our tree structure yields much better performance than a minimum spanning tree (MST) or a maximum computation tree (MCT). We apply the Divisible Load Theory (DLT) to recruit nodes into the resource tree until real-time constraints of the jobs are met. DLT [19], [8] is a well established theory in parallel computing that attain optimal scheduling given that load is perfectly divisible. Since the load information on the nodes changes dynamically without updating other nodes, a resultant communication overhead is incurred in collecting information for efficient load sharing. Moreover, as a node can be potentially requested by multiple peers, protocols are developed for the actual resource commitment at a node. Considering the inherent unpredictability and ad hoc nature of P2P networks, random job arrival and node failure/join are also addressed in our scheme.

The organization of this paper is as follows. In Section II, we formulate the problem and describe the proposed P2P computing protocol. In Section III, we present the detailed approach of our strategy for building and load sharing on a multi-level resource tree of maximum efficiency. Simulation specification and results are presented in Section IV and the paper is concluded in Section V with future extensions.

II. Proposed Framework for P2P Computing

In a real-time P2P system, it is assumed that each node, besides processing its local workload, has spare computational power to share with other peers. Application tasks that demand external computational power, can originate from any node at any time with varying resource requirements and timing constraints. More specifically, the surplus power available on each peer varies and its availability time differs. Peers can join and leave arbitrarily without notice. To adapt to such varying circumstances, a decentralized online resource management mechanism and task scheduling strategy that requires only local information is an apt choice. In this paper, we assume that all nodes in the network can individually compute and communicate with other nodes concurrently. However, each node can only communicate with one neighbor at a time. This means, a node has to send the communication messages sequentially, which adds up the total communication cost as it tries to recruit more peers. Similarly, jobs are sequentially dispatched to peers from the same parent and it takes different times depending on the amount of transfer and the communication bandwidth between the nodes.

In order to build a scalable and reliable load sharing mechanism, we adopt a decentralized resource location and job allocation scheme whereby each node only interacts with its direct neighbors. When a node P_x is overloaded, it sends a request to all its directly connected peers for their idle resources. The neighbors respond to P_x with their availability and reliability levels (credit). Upon receiving the feedback information, the local scheduler on P_x chooses the one that benefits the total computation capacity the most. If additional computing power is needed, a new node will be chosen starting from the nodes that have already been recruited, following the same criteria. This procedure will be repeated until the computational requirements are met and thereby gives rise to a multi-level resource tree rooted at P_x. Also, note that several nodes may be simultaneously trying to share their workload, which may give rise to multiple resource trees with different roots existing in the network. Hence, it is also possible that a particular node can be included in multiple resource trees at the same time.

Choice of peers at each step not only depends on the absolute computational power that is available on a node but also on the communication cost incurred to transfer the workload. Eventually it is the combination of the computational power and communication bandwidth together that determines the overall performance. When a powerful node is connected via an extremely slow link or when a fast link connects to a node that is barely available, it is desired not to include them in P2P computing. As a result, blindly including nodes that has either maximum computational power or minimum communication cost may not maximize the overall benefits. Further, even if a homogeneous network is considered, resource trees of different structures can also vary greatly in terms of their performance. Due to the sequential communication pattern, resource trees that are either too deep or too wide are obviously not preferred. Thus, identifying the most effective nodes with a balanced underlying tree structure from among the vast number of peers on the network to achieve maximum benefit is the essence of our approach. In the following sections, we describe in detail our proposed P2P computing model and the supporting communication protocol. The exact procedures for building and scheduling such a multi-level resource tree are given in Section III.

A. Peer Attribute

A node, P_i, is identified with the following attributes:

- Available computational power w_i (byte/sec).
- Credit c_i: this parameter reflects the reliability of a peer. When a new peer joins, it is included with an initial credit $c_i = c_0$. Whenever P_i consumes surplus power from other peers, c_i is deducted in proportion to the power w_b and time duration T_b that is “borrowed”. On the other hand, bonus points are credited to c_i proportionately to the power that c_i “lends” to its peers, under the condition

that the committed power is not withdrawn during processing. Otherwise, P_i will be penalized with its credit for abruptly breaking the initial power commitment.
- Credit minimum requirement γ: this is the minimum credit requirement for a peer to be allowed to use the P2P system. Peers with credit less than this threshold γ will be barred from using the idle computational power in the system. γ is defined as a system wide parameter to bar peers that always "borrows" but never "lends".

B. Job Attribute

A job, J_k, is characterized by the following attributes:

- Job size: S_k
- Execution deadline: D_k
- Reliability tolerance τ_k: this is a user specified reliability threshold for choosing the peers for a particular job. Peers with credit c_i under the threshold τ_k are considered untrustworthy and hence may not be considered to process job J_k for local reliability/QoS concerns.

C. Peer Behavior

Each node, P_i, has two possible modes:

- *Active mode*: Peers are in this mode so long as their credit is above the threshold γ. Peers in this mode are not only allowed to donate its surplus power with its neighboring peers but also to "borrow" idle power from other nodes. However, in order to stay in this mode, peers are forced to maintain a balanced behavior on its power consumption (decrease of c_i) and contribution (increase of c_i). Since extra penalty is inflicted for any "faithless" behavior, peers in this mode are generally considered reliable.
- *Passive mode*: Otherwise, as a penalty, peers will be downgraded to the passive mode when its credit falls below the threshold γ. Two types of inferior behaviors can actually cause a peer to fall into this mode. When a node consistently consumes more power than its contribution, its credit will eventually drop below the threshold which in turn will prevent it from further using the system. Such nodes are considered "selfish" and are mandatorily required to contribute enough before they are allowed to use the system again. On the other hand, even if a node does not demonstrate a consumption dominant behavior, its credit can still drop below the threshold when the node frequently cutoff its committed power supply to other peers. However, this penalty can be easily repaired by the node if it contributes its power constantly for some time without intermittent cutoff. This mechanism, essentially gives the node an opportunity to rebuild its reputation and the level of reliability. As a result, its credit level can be gradually recovered and eventually the node will be brought back to the active mode.

D. The Protocol to Select Peers

- A node, P_x, that demands extra computational power sends a *request message* to all of its immediate neighbors for their current status.
- A node, P_i, upon receiving a request message, checks its availability, and replies via a *status message* that contains its available surplus power w_i and credit c_i.
- Among all the neighbors that responded, P_x selects the one that maximize the total power enhancement based on a combined evaluation of the communication bandwidth and computational power. Consequently, a *select message* will be sent to the selected node, say P_j, to obtain its further commitment. The detailed peer selection policy will be given in Section III.
- Once the selected node P_j receives the select message, P_j, if not previously committed to other nodes, commits its availability to P_x via a *commit message*. Otherwise, a *busy message* will be sent to P_x if it happens that P_j has already committed to another node. In our protocol, peers commit their power based on a First Come First Serve policy.
- If a commit message is received from P_j, P_j will be attached onto the tree. Otherwise, if a busy message is received, another node P_k will be selected from among the remaining neighbors following the same criteria. As an alternative, P_k will be included in the tree if available.
- Once a new node is identified and included in the tree, P_x checks if the total power of the tree is sufficient. If the surplus power gathered is insufficient, a second round recruitment procedure will be initiated to seek for additional power. In order to find the next optimal node to be included, each node on the current tree will identify a local optimal candidate following a similar procedure as described above without carrying out the final commitment step. Further, among all these candidate nodes, the one that brings in maximum power benefit will be chosen. The concept of finding the equivalent power of a subtree, as will be elaborated in Section III, is used to facilitate such a peer selection procedure.
- The above procedure will be repeated till sufficient computational power has been gathered. Starting from the root, jobs will be dispatched down the tree, level by level according to the optimal load distribution as suggested by the scheduler.
- In the case of a node failure, the parent node will be notified with a *failure message*. Extra power will appropriately be located to make up the loss.

E. Divisible Load Theory

The load sharing protocol, described above, is highly dependent on the efficiency of the local scheduler to determine the load distribution among the peer nodes. It is well known that the total execution time of a job is minimal when all the processors executing different portions of the job finish execution at the same time. The Divisible Load Theory (DLT) [8], [19] develops scheduling assuming that the workload is perfectly divisible for such distribution among the processors, so that all processors finish at the same time. It considers heterogeneous processors and different communication times to send data to those processors. DLT is particularly suitable for data parallel operations, where the volume of data can be perfectly distributed without causing any error. DLT is highly applicable in parallel processing of many applications [19]. We have also demonstrated that DLT can be applied to packet processing in a network with good accuracy even though a packet is not divisible [10]. However, its possible use in P2P computing has not been explored earlier.

In this paper, we develop a scheduling theory at the root node based on DLT. However, instead of simply scheduling among the available directly connected peer nodes, we recruit more nodes in the form of a multiple level tree until the real-time requirements of the jobs at the root are satisfied. We consider the protocol overhead to establish the tree, as well as the communication time to dispatch the jobs. We also obtain many interesting sensitivity results indicating various computation-communication tradeoffs in Internet computing.

III. Design Strategy

In this section we describe the procedure of acquiring resources from a large network of peers, once the job requirement exceeds the capability of the local computer. The scheduler running at each individual node is only aware of its immediate peers, enquires them for their surplus power, and correspondingly schedules the workload upon job arrival.

A. Resource Analysis

A real-time job J_i is characterized by its arrival time (A_i), execution deadline (D_i), and size (S_i). A job is considered successful if it can be executed before its deadline. Assuming that the scheduler is invoked immediately at the time J_i arrives, the minimum computational power w_i^{min} needed to meet its deadline is $w_i^{min} = \frac{S_i}{D_i - A_i}$ neglecting the overhead incurred by the scheduler. Thus, a resource tree that offers no less than w_i^{min} computational capacity is needed to accommodate J_i successfully. On the other hand, once P_j commits its surplus power to a node P_k, its surplus power w_j will be made unavailable to the rest of the network until P_k releases its control of w_j. However, even if a peer is not capable of donating any computing cycles actively, it can still be involved in the resource discovery process and help transferring the workload to other neighboring nodes in a passive mode. Therefore, nodes recruited may consist of peers with heterogeneous power and communication bandwidth, and may be available at different times. A proper resource management and load scheduling algorithm is needed to guarantee both good load balancing and resource utilization.

B. Optimal Load Sharing

Given a pool of peer nodes, the total computation capacity w_{eq} that is available relies on the underlying topology of how peers are connected and scheduled. An optimal load sharing mechanism will help maximize the network resource utilization and in turn minimize the absolute resource demand. Thus, we need to follow an optimal scheduling pattern for resource evaluation thus ensuring only necessary number of peer nodes to be involved for a minimized time interval. In this section, we shall demonstrate how to achieve such an optimal scheduling pattern that explores the maximum equivalent computation capacity w_{eq} of a given resource tree.

1) Load Sharing on a Single-level Tree: For a particular job J_i, in order to build a resource tree of the right size, we need to determine whether the total power offered by a resource tree is sufficient to meet its requirement w_i^{min}. This estimation will be inaccurate if arrived just by adding up their computational power, because their communication times are different on the tree. Therefore, estimation with proper partitioning and scheduling is needed in order to determine the execution time of the job. We shall follow two optimality criteria to guarantee minimum execution time. First, as suggested by the Optimal Sequencing Theorem [8], the sequence of load distribution by the root node should follow a decreasing order of the communication speeds. Second, we partition and distribute the workload in such a way that all the participating peers stop computing at the same time. Consider a single-level heterogeneous resource tree rooting at P_x shown as the first level of the tree in Fig. 1(a), its optimal load distribution pattern is illustrated in Fig. 1(b). P_x divides the total workload into $(m+1)$ parts, keeps its own share α_x, and distributes the other fractions to the corresponding child node one after another in the decreasing order of their bandwidth. Each child node P_j starts processing immediately upon receiving its load fraction α_j and continues to do so until this fraction is finished.

Such a pattern, takes into consideration both, the communication and computation costs for sharing the workload. The following recursive equations are obvious from the timing diagram shown in Fig. 1(b). Note that w_j is the available computational power of P_j and $z_{x,j}$ represents the communication bandwidth between P_x and P_j.

$$\frac{\alpha_x}{w_x} = \frac{\alpha_1}{z_{x,1}} + \frac{\alpha_1}{w_1}, \tag{1}$$

$$\frac{\alpha_j}{w_j} = \frac{\alpha_{j+1}}{z_{x,j+1}} + \frac{\alpha_{j+1}}{w_{j+1}},\ j = 1, ..., m-1 \tag{2}$$

Using the normalizing equation $\alpha_x + \sum_{j=1}^{m} \alpha_j = 1$, we can solve these equations by expressing α_x and α_j in terms of α_m:

$$\alpha_x = w_x(\frac{1}{z_{x,1}} + \frac{1}{w_1}) \cdot \prod_{k=1}^{m-1} w_k(\frac{1}{z_{x,k+1}} + \frac{1}{w_{k+1}}) \cdot \alpha_m \tag{3}$$

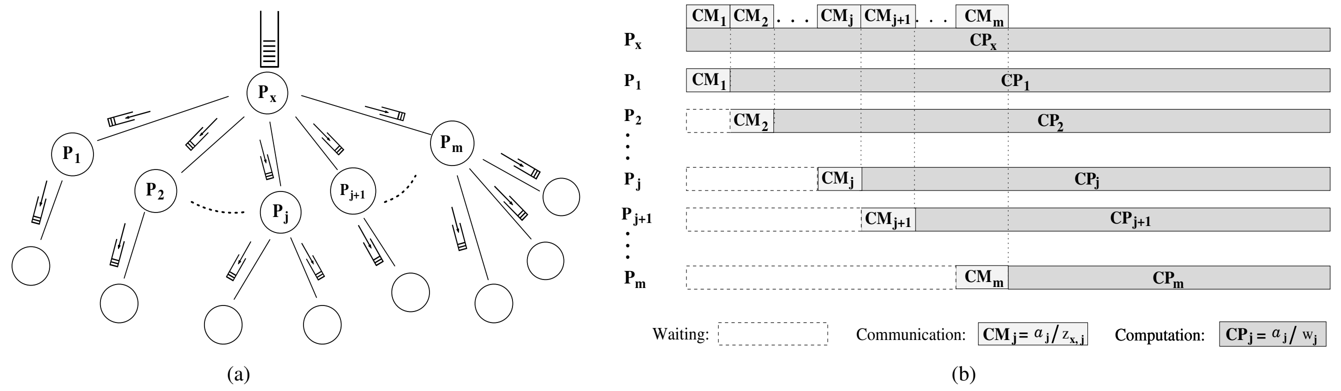
$$\alpha_j = \prod_{k=j}^{m-1} w_k(\frac{1}{z_{x,k+1}} + \frac{1}{w_{k+1}}) \cdot \alpha_m,\ j = 1, ..., m-1 \tag{4}$$

where,

$$\alpha_m = \frac{1}{1 + w_x(\frac{1}{z_{x,1}} + \frac{1}{w_1}) \prod_{k=1}^{m-1} f_k + \sum_{j=1}^{m-1} \prod_{k=j}^{m-1} f_k} \tag{5}$$

$$f_k = w_k(\frac{1}{z_{x,k+1}} + \frac{1}{w_{k+1}}) \tag{6}$$

Thus, using the above closed-form solution, we can obtain the optimal load fractions α for scheduling the workload on a single-level resource tree network. The desired scheduling pattern can be guaranteed so long as we follow the derived load fractions α_j on dispatching the load. Assuming the jobs



Px
P1
P2
Pj
Pj+1
Pm
CM1 CM2 . . . CMj CMj+1 . . . CMm
CPx
CP1
CP2
CPj
CPj+1
CPm
Waiting:
Communication: CMj = αj / zx,j
Computation: CPj = αj / wj
(a)
(b)

Fig. 1. Optimal Load Sharing Pattern on a Resource Tree

are perfectly divisible, the workload to be dispatched to a peer P_j for job J_i is $S_i \cdot \alpha_j$. As all the peers are guaranteed to stop processing at the same time, the total processing time is minimized and can be easily calculated by $T_{min} = S \cdot \frac{\alpha_x}{w_x}$.

2) Load Sharing on a Multi-level Tree: In the previous section, we designed an optimal load distribution strategy for a single-level resource tree. However, we may have to seek more levels of nodes in order to obtain sufficient computational power. Hence, it is crucial to extend the same optimal scheduling on a multi-level resource tree shown in Fig. 1(a). We use the concept of *equivalent node* to estimate the computational power available on a tree, which exhibits the same execution time as the original tree for a given workload. We determine the equivalent processing power w_{eq} of a single-level subtree as follows.

$$T_{min} = \frac{S \cdot \alpha_x}{w_x} = \frac{S \cdot 1}{w_{eq}} \tag{7}$$

$$w_{eq} = \frac{w_x}{\alpha_x} \tag{8}$$

Thus, for any given single-level tree, we can replace it with such an equivalent node with equal computational power. By traversing from the bottom level to the root of the tree, we can simply replace each single-level subtree with its equivalent node till the root is reached. In this way, the entire tree can be reduced to a single equivalent node P_{eq} with w_{eq} equivalent power. During this procedure, for every single-level subtree that come across, we obtain the local optimal load fractions α_i as suggested by (3) to (6) and use (8) to calculate its equivalent power. By following these load fractions hierarchically on each single-level subtree as and when we actually dispatch the workload on the resource tree, the optimal load distribution pattern as shown in Fig. 1(b) will be automatically guaranteed across the entire tree, both locally and globally.

C. Building the Resource Tree of Maximum Efficiency

In this section, we describe how a multi-level resource tree is constructed to fulfill the local job requirement. Consider a node P_x, which is unable to accommodate its own job J_i and is seeking extra computational power from the network. In order to accommodate J_i successfully, it is necessary to construct a resource tree Σ such that its equivalent computational power w_{eq} is a minimum of w_i^{min}.

Taking into consideration both the communication and computation costs, we use the concept of equivalent power w_{eq} as the node selection criteria. In this way, the tree is inherently constructed to maximize the overall performance based on the optimal load scheduling derived. Such a tree is termed a maximum efficiency tree (MET) as it represents the most effective set of peers for load sharing at the root. For any job failed to be accommodated locally, the local scheduler at that node will try to build a resource tree to execute the job. In this case any peer can be considered as the root node. Therefore, it is possible that one node may host multiple resource trees for separate jobs and further, multiple trees rooted at different nodes can exist on the network simultaneously. We thereby allow for a particular node to be conditionally shared by multiple trees ensuring no resource conflicts.

Thus, starting from the root, we shall include new nodes to the tree one at a time until the required computational power is gathered. From among all the reachable candidate peers of the current tree, the node that maximizes an increase in the total equivalent power of the tree (Δw_{eq}) will be chosen. Each time a new peer is attached to the tree, we calculate the updated equivalent power of the entire resource tree to see if it meets the requirement. Else, another round of node selection will be initiated starting from the new tree. As always, such a procedure can be carried out hierarchically on the tree and repeated till the capacity of the tree reaches the desired amount. As the nodes are selected following the optimal load sharing mechanism which considers both peer computation and communication capacities, the resulting resource tree is guaranteed to be balanced in shape, small in size and efficient in power. Fig. 2 presents the detailed procedure that should be followed to build such a tree.

D. Fault Tolerance Mechanism

In P2P networks, individual nodes may fail or malfunction at random, thereby are unable to provide the desired service. Jobs may subsequently fail due to unexpected node failures on the resource tree. To prevent such job failures, we can either find additional power backup before the actual node failures or alternatively devise after-failure resource make-up plans. In our scheme, the parent of the failed node is responsible for locating extra power to compensate for any power loss

```
Procedure: MET(...)
Job analysis: $w_i^{min} = \frac{S_i}{D_i - A_i}$, $\forall J_i$
Initial state: $\Sigma = \{P_x\}$, $w_{eq} = w_x$
if ($w_{eq} \geq w_i^{min}$), exit;
Do {    $\forall P_k \in \Sigma$ {
            $P^k_{candidate} = Find\_Candidate(P_k)$;
            $w^k_{candidate} = Find\_weq(\Sigma \cup P_{candidate})$;
        }
        $w^c_{candidate} = \max\{w^k_{candidate}\}$;
        $p_{next} = P^c_{candidate}$;
        $w_{next} = w^c_{candidate}$;
        $\Sigma = \Sigma \cup P_{next}$;
        $w_{eq} = w_{next}$;
} while ($w_{eq} < w_i^{min}$)
```

Fig. 2. Building Maximum Efficiency Trees

caused by its child nodes. We assume that nodes must notify its parent before leaving. Upon being notified with such a node failure, the parent node will follow the same tree construction procedure as described in Fig. 2 to build a supplemental subtree that is equivalent to the failed node in terms of its computation capacity. If enough power can not be gathered at this level, the node at the next upper level in the resource tree will be informed to further seek additional power. Such a procedure will be repeated till sufficient power is located or the root node is reached.

IV. Performance Evaluation

In this section, we present a detailed simulation based evaluation and discussion on the proposed load sharing framework. The simulator works in a decentralized fashion and the schedulers residing on individual nodes are triggered by the corresponding control messages as specified in the proposed protocol in Section II-D. GT-ITM Transit-Stub (TS model) [22] is used to generate the network topologies for our simulation. TS models the networks using a two-level hierarchy of routing domains, with transit domains interconnecting the lower level stub domains. By default, the latency of intra-transit domain links, stub-transit links and intra-stub domain links are set to $20ms$, $5ms$ and $2ms$ respectively [18]. Assuming a standard Ethernet packet of $1.5KB$, the corresponding latency generated above suggests a communication bandwidth of $0.6Mbps$, $2.4Mbps$ and $6Mbps$ respectively. In all the experiments, if unspecified, the default number of nodes in the simulated network is 5000 [18] and the average degree of the graph is 17.5, as given in the TS model. The simulator is designed to accommodate jobs with varying lengths and time constraints. Jobs arrive as a Poisson distribution with arrival rate $\lambda = 1/sec$ and the size of the job is determined by an exponential distribution $\beta = 50MB$. We generate the job deadlines arbitrarily with a uniform distribution $(100sec, 500sec)$. The average time to process a $50KB$ job on a peer at its full potential is configured as $60ms$. The above parameters are taken from our measurements while executing multimedia tasks in a Pentium-based cluster [21].

Control overhead of the load sharing algorithm is also simulated in our experiments. The total overhead is estimated based on the number of control messages generated to build the resource tree. According to our proposed protocol, the size of a control message is no larger than 20 bytes. Hence, for each control message, we consider a constant software overhead of $40\mu s$ for UDP as reported in [23] and a transmission overhead as determined by the corresponding communication bandwidth. In the following sections, we study the performance of our algorithm against various network and workload configurations. Effects of varying the job size, network bandwidth and node failure rate are observed. We compare the performance of MET with two other classic tree construction algorithms as follows.

- *Minimum Spanning Tree (MST):* On building the resource tree, at each step, the node with minimum communication cost is always chosen.
- *Maximum Computation Tree (MCT):* On building the resource tree, at each step, the node with maximum computational power is always chosen.

For a particular job, resource trees built using different node selection policy are equivalent in terms of their total computation capacity but can vary greatly in terms of their efficiency. To highlight the effectiveness of our algorithm MET over the MST/MCT algorithms, we identify the following metrics and observe their variation. Essentially, a good load sharing algorithm should demonstrate maximum success rate and utilization using minimum resources. We expect a tree to be more efficient than its equivalent if it requires fewer number of nodes and less amount of total computational power $\sum w$.

- *Resource utilization*: average fraction of time that all the nodes in the resource tree are kept busy. It indicates the effectiveness of our resource tree construction.
- *Success rate*: the percentage of jobs that are accommodated to meet their deadlines. Based on our theory, a 100% success rate can be guaranteed so long as resource trees can be constructed to supply the desired computational power. However, under certain workload intensity and network conditions situations arise when sufficient computational power fail to be located.
- *Average tree size*: average number of nodes recruited on the tree.
- $\sum w$: the total amount of power recruited per job.

A. Effect of Varying Job Size

In this section, we examine the impact of job size on the performance of our algorithm. Network topology is devised assuming an average of $5Mbps$ communication bandwidth. By increasing the job size, we actually raise the resource requirement to accommodate a job prior to its deadline and thereby larger resource trees are needed. This is equivalent to having a tighter deadline without changing the job size. As shown in Fig. 3, more nodes are included in the resource tree when the job size increases. As an immediate result, the resource utilization drops down due to larger communication cost incurred to dispatch the job. This effect can be observed for all three algorithms. However, MET demonstrates higher stability and consistently outperforms MST and MCT for both success rate and resource utilization. Yet, as shown in Fig. 3(c) and (d), MET always recruits the fewest number of nodes and demands least power. Essentially, this is due to the optimal load sharing mechanism adopted in MET that considers both computation and communication costs for selecting the peers. On the other hand, the peer selection policy of MST and MCT only favors either node communication or computation capability, thus demonstrating inferior performance due to their biased decision on selecting the nodes. Correspondingly, MST demands the most number of nodes due to its negligence of peer computational capabilities when selecting the nodes. Even though, except having a larger tree size, MST outperforms MCT in general as higher communication bandwidth on its trees ensures better resource utilization. MCT, on the other hand, ended up with recruiting much more power than the other two algorithms as it blindly includes nodes with maximum available power without proper utilization.

B. Effect of Varying Network Bandwidth

Due to the communication overhead incurred for building the resource tree and dispatching the workload, network bandwidth is an important parameter to evaluate the effectiveness of different algorithms. As the dispatching of the workload from a parent node to its child nodes is modeled sequentially, the higher the bandwidth the better the resource utilization and performance. As illustrated in Fig. 4, when the network bandwidth drops below $3Mbps$, the resource utilization for all three algorithms is affected due to the limited communication rate, thus resulting in less than 100% success rate. However, MET still manages to maintain a high success rate and much better resource utilization compared to MST and MCT. When the network bandwidth is increased up to $4Mbps$, a resource tree satisfying our need can be easily formed and the resource utilization of MET is guaranteed thereafter. Again, the average tree size and power demand for MET is the minimal as compared to MST and MCT, which indicates the efficiency of MET for constructing the tree.

Interestingly, MCT needs to recruit more number of nodes than MST when the bandwidth drops below $3Mbps$ as shown in Fig. 4(c). This is due to the fact that communication cost becomes the bottleneck while selecting the peers as the bandwidth decreases. Gradually when bandwidth recovers, MST recruits more nodes than MCT since computational capability of the peers regains its importance.

C. Performance Under Node Join/Failure

In order to examine the impact of node failures on the performance of our algorithm, nodes are failed at a constant rate μ following an exponential distribution. Peers are randomly selected to fail with probability correlated to their degree. The number of nodes in the network is kept roughly constant by matching the node arrival and failure rates. New nodes are added to the existing network following the same criteria as specified in GT-ITM. By varying the frequency of node failures, we are able to demonstrate the robustness of our fault tolerance mechanism. As shown in Fig. 5(a), MET is able to maintain a satisfying success rate even when node failures occur every second on average. Only when the failure rate is set to be as high as $8/sec$, the success rate starts to drop moderately. However, as shown in Fig. 5(b), larger failure rate incurs the average tree size to increase as more nodes are needed to compensate for the power loss.

D. Overhead Analysis

In this section, we study the effect of algorithm overhead by varying the job size and node failure rate. The overhead is measured in terms of the number of control messages sent to the network for building the resource tree. Different from the communication overhead incurred from dispatching the jobs (data messages), the delay incurred by control messages are purely algorithm based. As can be seen in Fig. 3(c), when the job size increases, more nodes are recruited in the resource tree to satisfy a greater need of computational power. Thus, when the size of the tree increases, more control messages are generated for building the resource tree as shown in Fig. 5(c). Such an overhead increase, has a direct impact on the efficiency of our load sharing algorithm. The longer it takes to construct the resource tree, the longer the resources wait for job execution, thereby resulting in a lower resource utilization as illustrated in Fig. 3(b).

Similarly in Fig. 5(d), node failure rate is varied to observe similar effects. As can be seen, the number of control messages increases as the failure rate increases. Though the size of the total workload remains the same in this experiment, greater computational power is needed to compensate for the power loss due to random node failures. As a result, when the failure rate increases, the actual number of nodes on the tree increases (as shown in Fig. 5(b)) and the overhead incurred to build the tree also increases.

V. Conclusion

In this paper, we proposed an efficient open P2P cycle sharing infrastructure Maximum Efficiency Tree (MET) that seeks to harvest computer cycles from ordinary users in an open access, non-institutional environment. The proposed scheme is capable of sharing real-time jobs among a pool of heterogeneous peer computers to ensure both load balancing and high resource utilization. The design of MET was based on Divisible Load Theory (DLT) applied to multi-level resource trees constructed dynamically to meet the job deadlines. The selection of peers is based on a joint evaluation of peer computational availability, reliability and communication bandwidth. Extensive sensitivity results have shown that MET consistently outperforms two other algorithms and satisfies both load balancing and real time constraints for job executions. Future works include implementing the protocol via PlanetLab [17] with proper result validation mechanism in place. Also, it would be interesting to extend the algorithms to handle the case while the workload is nondivisible.

References

[1] SETI@Home, *http://setiathome.ssl.berkeley.edu*
[2] Napster, *http://www.napster.com*

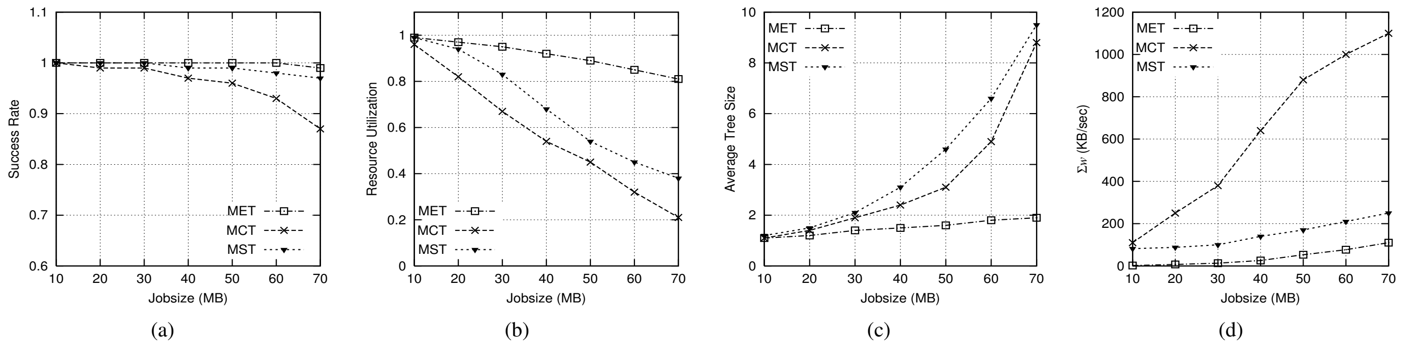


Fig. 3. Performance on Varying the Job Size

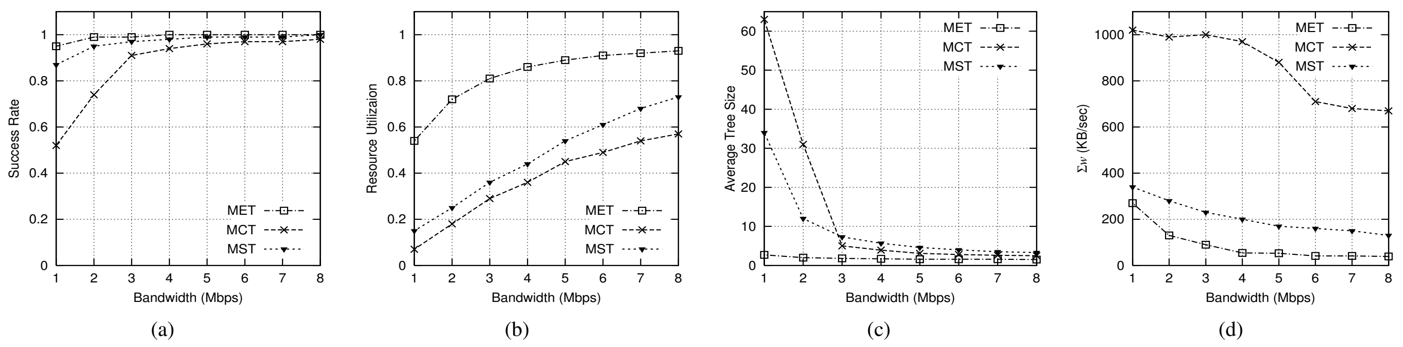


Fig. 4. Performance on Varying the Bandwidth

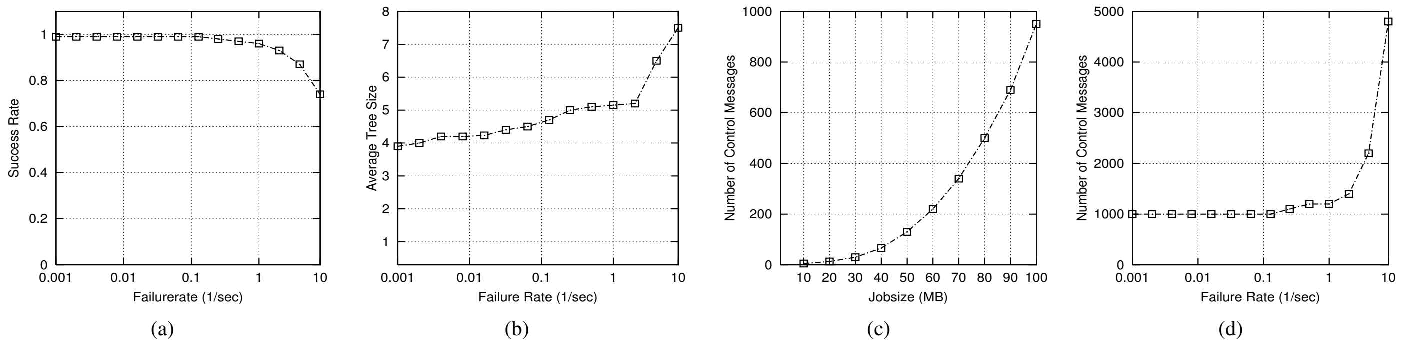


Fig. 5. Performance Robustness and Overhead Analysis

[3] Gnutella, *http://www.gnutella.wego.com*

[4] Avaki, *http://www.avaki.com*

[5] Condor, *http://www.cs.wisc.edu/condor*

[6] JXTA, *http://www.jxta.org*

[7] A. Awan, R. A. Ferreira, S. Jagannathan and A. Grama, "Unstructured Peer-to-Peer Networks for Sharing Processor Cycles", *Parallel Computing*, Vol. 32, Issue 2 , pp. 115-135, 2006.

[8] V. Bharadwaj, D. Ghose, V. Mani and T. G. Robertazzi, "Scheduling Divisible Loads in Parallel and Distributed Systems", *IEEE Computer Society Press*, Los Almitos, California, 1996.

[9] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of Supercomputer Applications*, 2003.

[10] J. Guo, J. Yao, and L. Bhuyan, "An Efficient Packet Scheduling Algorithm in Network Processors", *IEEE Infocom*, Miami, March 2005.

[11] A. Gupta, D. Agrawal and A. E. Abbadi, "Distributed Resource Discovery in Large Scale Computing Systems", *IEEE Symposium on Applications and the Internet*, 2005.

[12] D. Hughes and J. Walkerdine, "Distributed Video Encoding Over a Peer-to-Peer Network", *Proceedings of PREP*, Lancaster, UK, 2005.

[13] INTEL, "Peer-to-Peer-Enabled Distributed Computing", Intel White Paper, 2001.

[14] C. Liu, L. Yang, I. Foster and D. Angulo, "Design and Evaluation of a Resource Selection Framework for Grid Applications", *11th IEEE Symposium on High-Performance Distributed Computing*, 2002.

[15] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao, "Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet", *In The 3rd International Workshop on Peer-to-Peer Systmes*.

[16] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins and Z. Xu, "Peer to Peer Computing", HP Technical Report, HPL-2002-57, 2003.

[17] L. Peterson, T. Anderson, D. Culler and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet", *Proceedings of ACM HotNets-I Workshop*, Princeton, New Jersey, USA, October 2002.

[18] S. Ratnasamy, M. Handley, R. Karp and S. Shenker, "Topologically-Aware Overlay Construction and Server Selection", *IEEE Infocom*, 2002.

[19] T. G. Robertazzi, "Ten Reasons to Use Divisible Load Theory", *IEEE Computer*, Vol. 36, No. 5, pp. 63-68, May 2003.

[20] D. Tran, K. Hua and T. Do, "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming", *IEEE Infocom*, 2003.

[21] J. Yao, J. Guo, L. Bhuyan and Z. Xu, "Scheduling Real-time Multimedia Tasks in Network Processors", *IEEE Globecom*, Dallas, 2004.

[22] E. Zegura, K. Calvert and S. Bhattacharjee, "How to Model an Internetwork", *Proceedings of IEEE Infocom '96*, CA, May 1996.

[23] X. Zhang, L. Bhuyan and W. Feng, "Anatomy of UDP and MVIA for Cluster Communication", *Journal of Parallel and Distributed Computing, Special Issue on Cluster and Grid Computing*, 2005.