

# Exploiting Heterogeneity for Collective Data Downloading in Volunteer-based Networks<sup>\*</sup>

Jinoh Kim, Abhishek Chandra, and Jon Weissman  
Department of Computer Science and Engineering  
University of Minnesota, Twin Cities

4-192 EE/CS Building 200 Union Street SE, Minneapolis, MN 55455, USA  
{jinohkim,chandra,jon}@cs.umn.edu

## Abstract

*Scientific computing is being increasingly deployed over volunteer-based distributed computing environments consisting of idle resources on donated user machines. A fundamental challenge in these environments is the dissemination of data to the computation nodes, with the successful completion of jobs being driven by the efficiency of collective data download across compute nodes, and not only the individual download times. This paper considers the use of a data network consisting of data distributed across a set of data servers, and focuses on the server selection problem: how do individual nodes select a server for downloading data to minimize the communication makespan—the maximal download time for a data file. Through experiments conducted on a Pastry network running on PlanetLab, we demonstrate that nodes in a volunteer-based network are heterogeneous in terms of several metrics, such as bandwidth, load, and capacity, which impact their download behavior. We propose new server selection heuristics that incorporate these metrics, and demonstrate that these heuristics outperform traditional proximity-based server selection, reducing average makespans by at least 30%. We further show that incorporating information about download concurrency avoids overloading servers, and improves performance by about 17-43% over heuristics considering only proximity and bandwidth.*

## 1. Introduction

Volunteer-based distributed computing environments such as Grids and peer-to-peer systems have become increasingly popular over the past few years. The potential of volunteer computing for Grid systems has been established

by success stories such as SETI@home [2], that has reported aggregate CPU cycles on the order of 60 Tflops over several years. The increasing reliance of scientific and engineering disciplines on computation and harnessing huge data have led a number of projects into the Grid space, e.g. biomedical computing [4], high-energy physics [15], and health-care [9], to name a few. These domains have several distinguishing characteristics: their end-user communities are growing and rely on standard toolsets, these tools are often compute- and data-intensive, and the resource demands on these tools often outstrip what is available within any particular organization within a domain area. For example, BLAST (Basic Local Alignment Search Tool) [3] is a fundamental tool for the large bioinformatics community. BLAST queries require massive computation and data processing when operating against large databases, and the resources for BLAST could easily exceed a single investigator site.

A fundamental challenge for the deployment of services like BLAST on a volunteer Grid is the efficient distribution and dissemination of data to the computation nodes. For example, decomposing a BLAST query across a Grid typically requires that large databases (with sizes on the order of several Gbytes) be split up and sent to a large number of volunteer compute nodes to enable fast parallel execution. Such a requirement makes efficient data download crucial for the success of the end-to-end computation.

In this work, we consider the problem of concurrent downloading by a number of compute clients working on the same service request. This challenge is complicated by the extreme time-varying heterogeneity of the volunteer Grid as data servers have widely different capacity, bandwidth, and latency with respect to a downloading client. Simultaneous downloading from central data servers can lead to bottlenecks due to capacity and geographic constraints. Since worker nodes can be dispersed world-wide, the download times of some distant and poorly connected nodes

---

<sup>\*</sup>This work was supported in part by National Science Foundation grant ITR-0325949.

might overwhelm the overall execution time of the service request. Replicating to a few data servers can achieve more efficient data access, but still suffers from the problems of scalability and fault-tolerance if static replicas are used. Caching at the compute workers can help, but it is not well-suited to handle the problems of node churn and dynamic data generation.

To address these problems, we assume that the data is highly replicated across a data network and that clients make local decisions to select a server for download. Because a service request is not complete until all individual workers complete their execution, minimizing the slowest data download is crucial for achieving high performance overall. We refer to the download time of the slowest node in the computation as the *communication makespan*. Minimizing makespan is a challenge due to the heterogeneity of the data servers and the possibility of communication load imbalance (if large numbers of concurrent workers happen to pick the same data server). In this setting, simple strategies such as minimizing round-trip time (RTT) do not work well.

We investigate this problem in the context of two distributed computing infrastructures: BOINC (Berkeley Open Infrastructure for Network Computing) [1]—a volunteer-based compute network, and Pastry [18]—a volunteer-based data network. BOINC is a pull-based system upon which SETI@home was based. In our context, compute nodes pull the distributed work associated with service requests. The compute nodes then fetch the needed data files from the Pastry network, a peer-to-peer DHT-based storage system. We propose and analyze server selection heuristics that can address the dynamic and heterogeneous nature of the Grid environment. The contributions of this paper are as follows:

- We show how the heterogeneity of data servers impacts concurrent downloading behavior.
- We quantify the impact of different metrics such as proximity, bandwidth, load and concurrency, on download time.
- We demonstrate that simple heuristics exploiting heterogeneity outperform typical proximity-based server selection.

## 2. System Model

In our system model, we incorporate a compute network as well as a data network to host large-scale services that operate on large datasets, require significant computation, and are accessed by remote end-users. An example is BLAST where a service request is split across many compute nodes that operate on portions of large read-only databases that

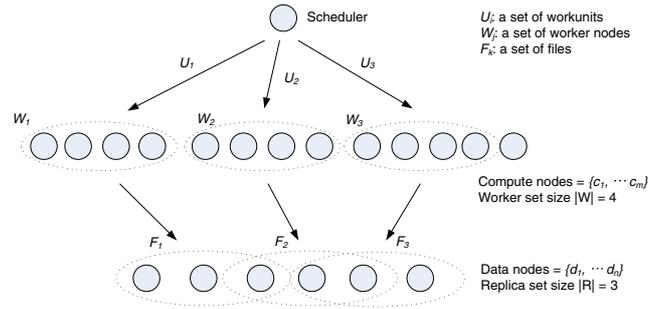


Figure 1. System model

they download. In our model, the *compute network* consists of worker nodes that provide CPU cycles for computation. The *data network* is responsible for transmitting data from the data generation or storage locations to the worker nodes that operate on the data. An example of a compute network would be BOINC [1] and a data network would be Pastry [18].

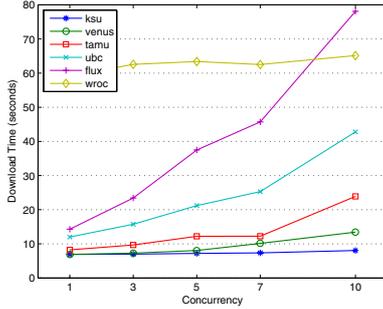
Formally, the compute network consists of compute nodes  $\{c_1, c_2, \dots\}$ , and the data network is composed of data servers  $\{d_1, d_2, \dots\}$ . All the data files required for computation are assumed to be replicated across multiple servers in the data network, with  $R_i$  denoting the set of replicas for a file  $F_i$ . For scheduling purposes, a job  $J^1$  is decomposed into a set of tasks or workunits  $U_i (0 < i \leq n)$ , where each  $U_i$  requires one (or more) files  $F_i$ . In BOINC, workunits are often redundantly scheduled to worker nodes to improve reliability. As shown in Figure 1, the scheduler assigns a workunit  $U_i$  to a set of worker nodes  $W_i$ , each of which then attempts to download the associated file  $F_i$  from one of its replicas.

To download the file, each node  $w_{ij} \in W_i$  queries the data network for a set of servers holding the file  $F_i$ , along with their current state. The server state might include attributes such as the server capacity, its roundtrip latency from the worker node, etc. We define a candidate set  $C_{ij}$  for the worker node  $w_{ij}$  to be the set of replicas that the data network returns in response to the query. The size and composition of the candidate set is a function of the degree of replication, the time-out values used to search for replicas, and is dependent on the type of data network employed as well as the location of the worker node. The worker then uses a *server selection heuristic*  $H$  to select a server  $s_{ij}$  from the candidate set for the actual download. In other words, a server selection heuristic is a mapping function:

$$H : C_{ij} \rightarrow s_{ij}, \text{ where } s_{ij} \in C_{ij}.$$

Minimizing makespan is key as the service request will not be complete until all tasks are finished. Since data

<sup>1</sup>In our application model, a job is a service request.



**Figure 2. Heterogeneity of servers: concurrency refers to the number of simultaneous download requests. The data size is 2MB.**

download or communication is a key component of the workunit execution time, we define the *communication makespan* to be the maximal download time for a workunit  $U_i$ :

$$makespan = \max_{w_{ij} \in W_i} \{T_{ij}\},$$

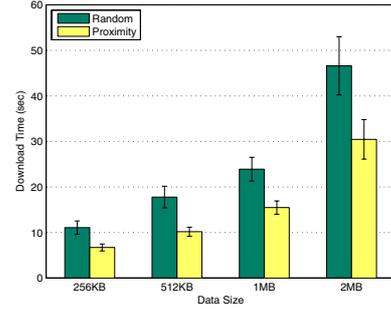
where,  $T_{ij}$  is the download time for worker  $w_{ij} \in W_i$ .

The objective of this work is to reduce the makespan by selecting good data servers. A challenge is that the individual compute workers are distributed and isolated from each other. Collecting global state dynamically to improve server selection is not scalable or practical. On the other hand, a greedy server selection technique might choose the best server for each node locally without consideration of the other workers. Figure 2 shows how such a greedy approach might degrade the download performance of servers by increasing concurrency of downloads. This experiment uses a set of PlanetLab nodes. Another point to be noted from this graph is the heterogeneity of nodes in PlanetLab—each server has a different level of sensitivity with respect to concurrent downloading requests, indicating the difference in their capacities. Our goal is to incorporate such server heterogeneity to do local server selection while avoiding poor global decisions.

### 3. Server Selection Heuristics

In this section, we investigate different metrics that affect the efficiency of data downloading. Based on the impact of these metrics, we present heuristics for selecting data servers in our environment. A key requirement of our model is to minimize the overall makespan of a service request, and not to simply minimize the individual download times at each worker independently.

Proximity has been employed as a network metric of choice in several domains ranging from routing in over-



**Figure 3. Performance comparison of random- and proximity-based selection.**

lay networks [18][16][21] to nearest server selection in the Internet [10]. In general, proximity refers to the network distance between hosts and can be measured in terms of roundtrip latency between the hosts, using TCP roundtrip times or ICMP echo packets. According to conventional wisdom, proximity is the dominant factor in predicting data download performance. As a result, proximity information is collected by many data network infrastructures. For example, FreePastry [11] is a DHT-based network that provides end-to-end roundtrip information between peers based on predicted latency values. Figure 3 compares mean download times with errorbars<sup>2</sup> observed by workers using proximity-based selection vs. random selection for different data download sizes. As can be seen from the figure, proximity-based selection always outperforms random selection, and hence, proximity information is clearly important. However, is proximity the only useful metric that determines the choice of a server for data downloading?

To answer this question, we conducted experiments on a 43-node PlanetLab slice to determine the various parameters that affect download performance. An experimental evaluation was performed on PlanetLab over a 7-month period (Apr-Oct 2006). Several measures are explored, and we find strong correlations not only between RTT and download performance but also between network bandwidth and download performance. RTT is gathered from our deployed data network, Pastry [18], while Iperf [12] statistics are used to determine network bandwidth. Figures 4(a) and 4(b) show the relationship of download times with RTT and bandwidth respectively. We use 4 different data download sizes: 256KB, 512KB, 1MB, and 2MB. Each point in these graphs corresponds to a single data download. We make the following observations from these graphs:

<sup>2</sup>Errorbars in this paper represent 95% confidence interval, unless otherwise noted.

- *Observation 1:* In the case of RTT (Figure 4(a)), the data download time for each data size is lower-bounded by a linear curve, indicating the presence of a near-linear relationship to RTT. However, the variation in the observed download times suggests the impact of other parameters.

- *Observation 2:* In the case of bandwidth (Figure 4(b)), we observe that the lower bound on the download time for each data size has an exponential relationship to bandwidth. In other words, servers with fairly large bandwidth (e.g., those over 10Mbps) do not show considerable difference among their download time trends, while low bandwidth servers (e.g., those under 1Mbps) show a sharp increase in the download time as the bandwidth decreases. However, again, the variation in the observed download times suggests the impact of other parameters.

- *Observation 3:* We also observed that system load and concurrency are correlated to download time (the effect of concurrency is illustrated in Figure 2)<sup>3</sup>. These factors may impact the performance if too many concurrent downloads occur from the same server simultaneously. Such concurrency may happen due to race conditions, where independent workers making independent download decisions might select the same “desirable” server, in turn overloading it. Such overloading should be avoided to minimize the communication makespan.

Based on these observations, we gain the following insights into making server selection:

- Servers with low bandwidth (e.g. under 1 Mbps) should be avoided, even if their RTT is small.
- Servers with relatively high bandwidth (e.g. over 10 Mbps) should be preferred, and should use RTT as a discriminator.
- Servers with medium bandwidth (e.g. between 1-10 Mbps) should be discriminated by load or concurrency.

We use these insights to derive a *cost function*  $f_{i,j}$  that is used by a worker  $i$  to quantify the desirability of a server  $j$  for data download:

$$f_{i,j} = \alpha_j \cdot rtt_{i,j}, \quad (1)$$

where  $rtt_{i,j}$  is the RTT between the worker and the server, and  $\alpha_j$  is a weight used to incorporate other server parameters, defined as follows:

$$\alpha_j = e^{(k_j/bw_j)}, \quad (2)$$

where  $bw_j$  is the bandwidth of the server, and  $k_j$  is a (server-dependent) constant that incorporates parameters such as load and concurrency (discussed below).

<sup>3</sup>We did not find correlation to other parameters such as CPU power, size of memory, etc. in our experiments.

This cost function has the following desired properties based on our observations. First, the cost function is directly proportional to RTT (Observation 1), such that the proportionality constant is the weight  $\alpha_j$  which incorporates the effect of other server parameters. Second, the cost function has an exponential relation to the server bandwidth (Observation 2). Finally, we define the constant  $k_j$  to incorporate factors such as load and concurrency (Observation 3). Note that the values returned by the cost function are not meant to be absolute (i.e., these values are not used for predicting the actual download times), but their relative values can be used for ranking multiple servers in the order of their selection desirability.

We define three heuristics for server selection that use different values for  $k_j$ :

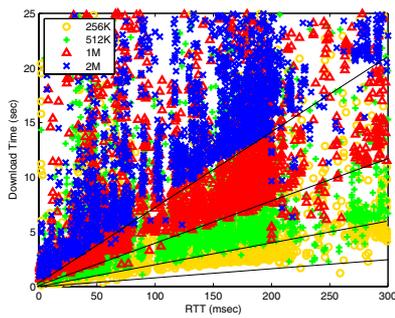
- **BW-ONLY:** Uses  $k_j = constant$ . We use  $k_j = 1$  in our experiments.
- **BW-LOAD:** Uses  $k_j = load_j$ , where  $load_j$  is the 5-minute average system load on the server.
- **BW-CAND:** Uses  $k_j = num\_cand_j$ , where  $num\_cand_j$  is the number of times the servers has responded as a candidate within the last 15 seconds.

The heuristic BW-ONLY uses only the RTT and the bandwidth metrics for selecting a server, while the other heuristics BW-LOAD and BW-CAND also use average system load and concurrency information respectively. For BW-LOAD, we use 5-minute system load as the load metric which is obtained by Linux/Unix `uptime` command. As the load value grows, the weight becomes large, and predicted download cost goes up. BW-CAND uses the number of times the server has responded as a candidate within a predetermined time window. In the experiments, we set the time window to 15 seconds which is equal to the search time we used in the DHT ring. Using the heuristic BW-CAND, servers which have responded as a candidate several times recently are penalized, because they are more likely to be selected by multiple workers, and hence to be concurrently serving data in the near future.

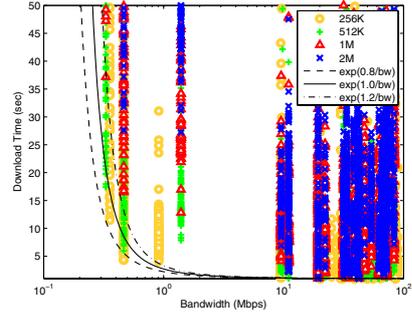
## 4. Performance Evaluation

### 4.1. Experimental Testbed and Methodology

To evaluate the various server selection heuristics described in the previous section, we conducted experiments on a set of randomly selected PlanetLab nodes geographically distributed across the globe: North America 20, Europe 19, and Asia/Pacific 4 nodes. For data replication and download, we implemented a data network over FreePastry [11], a public Pastry implementation developed by Rice



(a) Performance w.r.t RTT



(b) Performance w.r.t bandwidth

**Figure 4. Performance correlation to RTT and Bandwidth: Download performance has a linear relationship to RTT that is offset by secondary factors. In the case of bandwidth, download performance decreases exponentially as bandwidth goes beyond 1Mbps.**

**Table 1. Experiments setup**

Experiments	Nodes	Replication	Candidate	Concurrency	Data Size	Queries
EX-1	19	10	5	5	2M	690
EX-2	33	10	5	5	256K,512K,1M,2M	> 200
EX-3	29	10	5	5,10,15	2M	> 250
EX-4	29	10	5	5	256K,512K,1M,2M	> 450

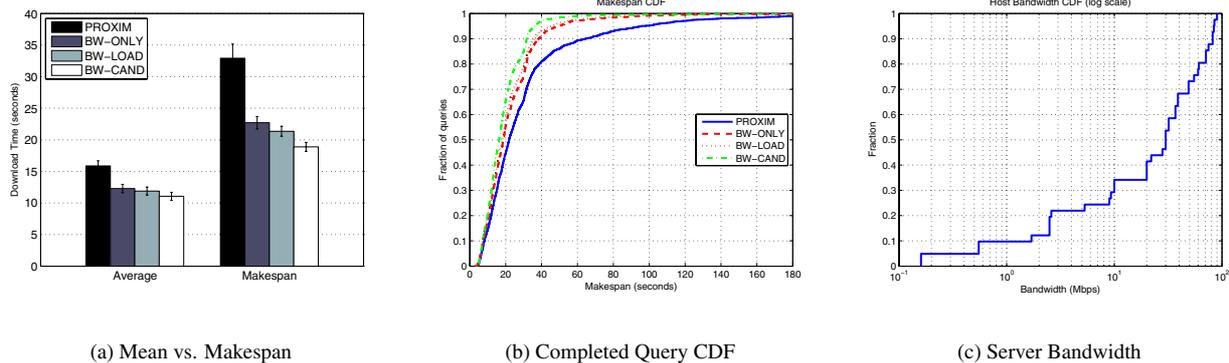
University. FreePastry provides the underlying data placement, request and data routing mechanisms. We conducted each of our experiments as follows: data files are distributed over the data network at the beginning of each experiment, and then data queries are generated for downloading these data files. For each data query, a set of worker nodes are selected randomly to request the same designated file concurrently. For fair comparison across the different server selection heuristics, queries are interleaved: e.g., each set of worker nodes downloads the files first with the PROXIM (proximity-based heuristic) selection, followed by the BW-ONLY selection, etc. Some queries might fail due to reasons such as churn (e.g., nodes going down) or query incompleteness (e.g., message routing failure in the DHT ring). If any query fails in the interleaved set of queries, the result is discarded in our analysis.

There are two main parameters that we vary across our experiments: (i) *data download size* ( $D$ ), with values of 256KB, 512KB, 1MB, and 2MB, and (ii) *concurrency* ( $C$ ) of client accesses for the same file, using values of 5, 10, and 15. We use a replication factor of 10 for placing each data file to provide us with a relatively large set of candidate servers for download. This allows a better comparison of server selection heuristics. In addition, we use different sets of machines in each experiment, with randomly chosen

data placement (driven by FreePastry) to generate different environmental conditions. Table 1 shows the various experimental scenarios we created. The scenarios differ in some of the parameters above, as well as the specific set and number of nodes that were used.

## 4.2. Comparison of Server Selection Heuristics

Figure 5 compares the various server selection heuristics for  $C=5$  and  $D=2MB$ , using the aggregated results of all the experiments that used  $C=5$  and  $D=2MB$ . Figure 5(a) plots the average download time and makespan respectively for the various heuristics. The first observation we make from the figure is that the bandwidth-based heuristics perform much better than proximity-based server selection in terms of both the average as well as the makespan. Moreover, the gaps in performance are larger in the case of makespan ( $\sim 30-45\%$ ) than in mean download time ( $\sim 20-30\%$ ). This result is also seen from Figure 5(b) that plots the CDF of the download completion times. As seen from the figure, 10% of PROXIM queries take more than 60 seconds to complete, while the bandwidth-based heuristics take less than 40 seconds to complete 90% of their queries. Moreover, these heuristics finish most of their queries within around 100



**Figure 5. Performance comparison of different heuristics ( $C=5$ ,  $D=2\text{MB}$ , Num Queries  $> 1800$ ): (a) shows the average download time and makespan, (b) shows the CDF of download completions, and (c) shows the bandwidth distribution of data servers.**

seconds, while about 5% percent of queries are unfinished for PROXIM selection. Thus, this result implies that *using bandwidth in addition to proximity produces better performance, not only in terms of individual download, but also in overall makespan.*

Another observation we make from Figures 5(a) and 5(b) is that BW-CAND shows the best results for both mean download time and makespan. In the case of makespan, BW-CAND gains over 40% compared to PROXIM, while BW-ONLY and BW-LOAD show 30-40% gains. Figure 5(b) shows the CDF of the completion times of all the queries. This result implies that *incorporating concurrency in addition to bandwidth improves the performance even further.*

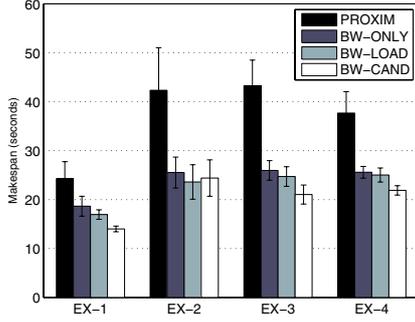
While Figure 5 shows the aggregated results, Figure 6 depicts the results separately for each experiment. Once again, we see that bandwidth-based heuristics outperform proximity-based server selection in all cases, and that BW-CAND performs best in all cases (except EX-2, where its performance is equivalent to the other heuristics).

The basic reason why the bandwidth-based heuristics outperform proximity-based selection is that they can exclude extremely slow servers. In our experiments, the participating hosts are almost uniformly distributed through the bandwidth ranges as shown in the Figure 5(c): nearly 10% of the hosts have a bandwidth under 1Mbps, 50% of the hosts have under 30Mbps, and upper 10% hosts have over 80Mbps bandwidth. By penalizing low bandwidth servers, the bandwidth-based heuristics can select servers with better bandwidth, even though they may be a little further from the worker node. Given that PlanetLab systems are well-organized compared to systems in typical volunteer-based networks, we anticipate that the heuristics can differentiate the results much more in such environments.

**Table 2. Server Bandwidth Distribution**

Class	Low	Medium	High
	$< 1\text{Mbps}$	$1 - 10\text{Mbps}$	$> 10\text{Mbps}$
EX-1	5%	26%	67%
EX-2	12%	6%	82%
EX-3	0%	24%	76%
EX-4	0%	24%	76%

The reason BW-CAND performs the best can be found in the bandwidth distribution of servers as shown in Table 2. Here, we classify hosts in three categories: low, medium, and high bandwidth, based on their bandwidth values. All of the bandwidth-based heuristics can penalize low-bandwidth servers (i.e. those with less than 1Mbps), but may not penalize medium-bandwidth servers (i.e. those between 1Mbps and 10Mbps). In fact, BW-ONLY might not penalize such medium-class servers because the weight value  $\alpha_j$  is likely to stabilize beyond 1Mbps, due to its exponential relation to bandwidth (Equation 2). In addition, if the average load is low on these medium-class hosts (close to 1), BW-LOAD also does not penalize them. In contrast, BW-CAND can penalize these servers, if too many clients try to select them, thus leading to higher values of recent candidate set queries. Thus, BW-CAND is able to provide better performance for such servers, by proactively preventing overloads from happening, while BW-LOAD is able to react only to past observed load. Unlike other experiments, EX-2 shows all the heuristics to have similar performance. This can be explained by the fact that EX-2 has only 6% medium-class servers (as seen from Table 2), whereas other experimental scenarios have more than 20% medium-



**Figure 6. Performance comparison for individual experiments ( $C=5$ ,  $D=2\text{MB}$ ): PROXIM is the worst and BW-CAND is the best in all cases except EX-2, where BW-CAND is comparable to BW-ONLY and BW-LOAD.**

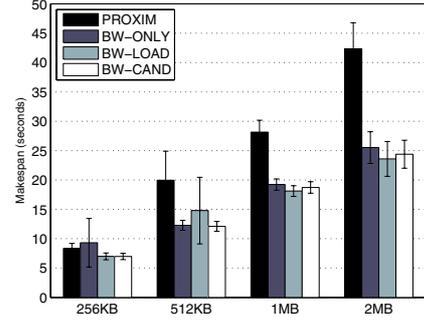
class servers, thus reducing the differentiation opportunity for BW-CAND. However, note that BW-CAND does not perform any worse than other heuristics even under these conditions.

### 4.3. Impact of Data Download Size

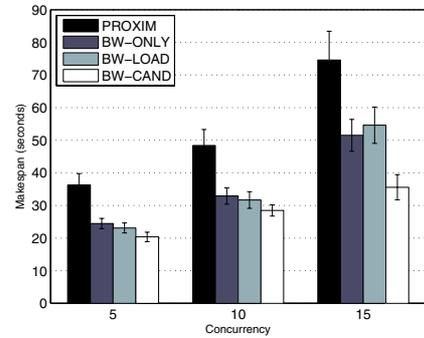
Figure 7 shows the average makespan obtained for varying data sizes from 256KB to 2MB. The figure indicates that the bandwidth-based heuristics outperform proximity-based selection much more significantly as the data size increases, going from 16% for  $D=256\text{KB}$  to 42% for  $D=2\text{MB}$  when comparing BW-CAND to PROXIM. Another experiment with varying data sizes (EX-4) showed similar results, and has been omitted due to space constraints. This result indicates that while proximity-based selection may be sufficient for small data sizes, *server bandwidth assumes an important role for larger data sizes*.

### 4.4. Impact of Concurrency

To see the impact of concurrent downloads for the same files, we used concurrency values of 10 and 15 in addition to the value of 5 used in our previous experiments. Since the replication factor for data placement is set to 10, race conditions would be unavoidable in this experiment with clients selecting the same server for download in several cases. In this case, we see that the bandwidth-based heuristics outperform proximity consistently. Moreover, we see that as the concurrency increases, BW-CAND starts outperforming the other heuristics, indicating that *avoiding overloads by reducing concurrent data downloads from the same server is important*.



**Figure 7. Impact of data size (EX-2;  $C=5$ ): The gaps between PROXIM and the other heuristics increase as the data size grows.**



**Figure 8. Impact of concurrency (EX-3;  $D=2\text{MB}$ ): BW-CAND widens the gap against other heuristics as concurrency increases.**

## 5. Related Work

Much server selection research has been conducted in the Internet [19, 14, 5, 10]. Carter and Crovella [5] also considered available bandwidth information (end-to-end link latency and congestion information) between the client and server for server selection. The major difference from our work is that we also consider system aspects due to heterogeneity and system loading, such as capacity and workload. In [10], the authors compared four metrics: previously measured RTT, IP path length, AS path length, and geographical distance. According to their results, RTT works best as a distance metric in the Internet to find the nearest server. Our results show that the nearest server does not guarantee better performance in terms of actual data transfer.

Estimating network proximity is a hot topic in peer-to-

peer networks. To estimate proximity, landmarks [13, 7], direct communication [8], and topological information [17] have been proposed. Our approach is to utilize the best communication metrics from the underlying infrastructure and would be compatible with all of these approaches.

BitTorrent is a peer-to-peer file distribution protocol which enables parallel downloading using network coding in peer-to-peer networks [6]. In [20], the authors suggest large data sharing using BitTorrent in computational Grids. The ultimate goal of this work is similar to ours. The major difference is that we are focusing on replica selection for downloading data blocks and accounting for heterogeneity.

## 6. Conclusion

In this paper, we examined the problem of efficient data dissemination for scientific computing in a volunteer-based network. We considered the use of a data network consisting of data distributed across a set of data servers, and focused on the *server selection problem*: how do individual nodes select a server for downloading data to minimize the *communication makespan*—the maximal download time for a data file. The communication makespan is an important measure because the successful completion of jobs is driven by the efficiency of *collective data download* across compute nodes, and not only the individual download times. Through experiments conducted on a Pastry network running on PlanetLab, we showed that conventional proximity-based server selection does not always produce good results. We demonstrated that nodes in a volunteer-based network are heterogeneous in terms of several metrics, such as bandwidth, load, and capacity, which further impact their download behavior. We proposed new server selection heuristics that incorporate these metrics, namely, the server bandwidth, load, and download concurrency, and showed that these heuristics outperform proximity-based server selection, reducing average makespans by at least 30%. We further showed that incorporating information about download concurrency avoids overloading servers, and improves performance by about 17-43% over heuristics considering only proximity and bandwidth.

## Acknowledgments

We thank Bret McGuire for offering testbed codes, and Seonho Kim for helpful comments on experiments.

## References

[1] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *Proc. of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10, 2004.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[3] BLAST: The basic local alignment search tool, <http://www.ncbi.nlm.nih.gov/blast>.

[4] caBIG: cancer biomedical informatics grid, <http://cabig.cancer.gov/index.asp>.

[5] R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical report, Boston University, MA, USA, 1996.

[6] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[7] M. Costa, M. Castro, A. Rowstron, and P. Key. Pic: Practical internet coordinates for distance estimation. In *International Conference on Distributed Systems*, 2004.

[8] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM'04*, pages 15–26, 2004.

[9] ePCRN: electronic primary care research network, <http://www.epcrn.org>.

[10] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. Idmaps: a global internet host distance estimation service. *IEEE/ACM Trans. Netw.*, 9(5):525–540, 2001.

[11] FreePastry, <http://freepastry.org>.

[12] PlanetLab Iperf, <http://jabber.services.planet-lab.org/php/iperf>.

[13] E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proc. of IEEE INFOCOM'02*, pages 170–179, 2002.

[14] T. Ogino, M. Kosaka, Y. Hariyama, K. Matsuda, and K. Sudo. Study of an efficient server selection method for widely distributed web server networks. In *10th Annual Internet Society Conference*, 2000.

[15] PPDG: Particle physics data grid, <http://www.ppdg.net>.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01*, pages 161–172, 2001.

[17] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically aware overlay construction and server selection. In *Proc. of IEEE INFOCOM'02*, 2002.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329+, 2001.

[19] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek. Selection algorithms for replicated web servers. *SIGMETRICS Perform. Eval. Rev.*, 26(3):44–50, 1998.

[20] B. Wei, G. Fedak, and F. Cappello. Scheduling independent tasks sharing large data distributed with bittorrent. In *Proc. of the 6th IEEE/ACM International Workshop on Grid Computing (GRID'05)*, 2005.

[21] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. In *IEEE Journal on Selected Areas in Communications*, 2003.