Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 07-028

Accessibility-based Resource Selection in Loosely-coupled Distributed Systems

Jinoh Kim, Abhishek Chandra, and Jon Weissman

November 20, 2007

Accessibility-based Resource Selection in Loosely-coupled Distributed Systems *

Jinoh Kim, Abhishek Chandra, and Jon B. Weissman
Department of Computer Science and Engineering
University of Minnesota, Twin Cities
Minneapolis, MN 55455, USA
{jinohkim,chandra,jon}@cs.umn.edu

Abstract

Large-scale distributed systems provide an attractive scalable infrastructure for network applications. However, the loosely-coupled nature of this environment can make data access unpredictable, and in the limit, unavailable. Availability is normally characterized as a binary property, yes or no, often with an associated probability. However, availability conveys little in terms of expected data access performance. Using availability alone, jobs may suffer intolerable response time, or even fail to complete, due to poor data access. We introduce the notion of accessibility*, a more general concept, to capture both availability and performance. An increasing number of data-intensive applications require not only considerations of node computation power but also* accessibility *for adequate job allocations. For instance, selecting a node with intolerably slow connections can offset any benefit to running on a fast node.*

In this paper, we present accessibility-aware resource selection *techniques by which it is possible to choose nodes that will have efficient data access to remote data sources. We have that the* local *data access observations collected from a node's neighbors are sufficient to characterize accessibility for that node. We then present resource selection heuristics guided by this principle, and show that they significantly outperform standard techniques. We also investigate the impact of churn in which nodes change their status of participation such that they lose their memory of prior observations. Despite this level of unreliability, we show that the suggested techniques yield good results.*

1 Introduction

Large-scale distributed systems offer the appeal of scalability for hosting network applications. This virtue has led to the deployment of several distributed applications in large-scale, loosely-coupled environments such as peer-to-peer computing [2], distributed storage systems [18, 24, 4], and Grids [9, 23, 22]. In particular, the ability of large-scale systems to harvest idle cycles of geographically distributed nodes has led to a growing interest in cycle-sharing systems [46] and @home projects [1, 39]. However, a major challenge in such systems is the network unpredictability and limited bandwidth available for data dissemination. For instance, the BOINC project [6] reports an average throughput of only about 289 Kbps, and a significant proportion of BOINC hosts shows an average throughput of less than 100 Kbps [2]. In such an environment, even a few MBs of data transfer between poorly connected nodes can have a large impact on the overall application performance. This has severely restricted the amount of data used for computation in such computation platforms, with most computations taking place on small data objects.

Emerging scientific applications, however, are data-intensive and require access to a significant amount of dispersed data. Such data-intensive applications encompass a variety of domains such as high energy physics [34], climate prediction [27], astronomy [3] and bioinformatics [5]. For example, in high energy physics applications such as the Large Hadron Collider (LHC), thousands of physicists worldwide will require access to shared, immutable data at the scale of petabytes [19]. Similarly, in the area of bioinformatics, a set of gene sequences could be transferred from a remote database to enable comparison with input sequences [42]. In these examples, performance depends critically on efficient data delivery to the

*This work was supported in part by National Science Foundation grant ITR-0325949.

computational nodes. Moreover, the efficiency of data delivery for such applications would critically depend on the location of data and the point of access. Hence, in order to accommodate data-intensive applications in loosely-coupled distributed systems, it is essential to consider not only the computational capability, but also the data *accessibility* of computational nodes to the required data objects. The focus of this paper is on developing resource selection techniques suitable for such data-intensive applications in large-scale computational platforms.

Data availability has been widely studied over the past few years as a key metric for storage systems [24, 18, 4]. However, availability is primarily used as a server-side metric that ignores client-side accessibility of data. While availability implies that at least one instance of the data is present in the system at any given time, it does not imply that the data is always accessible from any part of the system. For example, while a file may be available with 5 nines (i.e. 99.999% availability) in the system, real access from different parts of the system can fail due to reasons such as misconfiguration, intolerably slow connections, and other networking problems. Similarly, the availability metric is silent about the efficiency of access from different parts of the network. For example, even if a file is available to two different clients, one may have a much better connection to the file server, resulting in much less download time compared to the other. Therefore, in the context of data-intensive applications, it is important to consider the metric of *data accessibility*: how efficiently can a node access a given data object in the system. Note that accessibility depends upon the point of access, and not just the presence of a data object in the system. In this context, accessibility directly implies availability, but not vice versa.

The challenge we address is the characterization of accessibility from individual client nodes in large distributed systems. This is complicated by the dynamics of wide-area networks which rule out static a-priori measurement, and the cost of on-demand information gathering, which rules out active probing. Additionally, relying on global knowledge obstructs scalability, so any practical approach must rely on local information. To achieve accessibility-aware resource selection, we exploit *local*, historical data access observations. This has several benefits. First, it is fully scalable as it does not require global knowledge of the system. Second, it is inexpensive as we employ observations of the node itself and its directly connected neighbors (i.e. one-hop away). Third, past observations are helpful to characterize the access behavior of the node. For example, a node with a thin access link is likely to show slow access most of the time. Last, by exploiting relevant access information from the neighbors, it is possible to obviate the need for explicit probing (e.g. to determine network performance to the server), thus minimizing system and network overhead. Our key research contributions are as follows:

- We present accessibility estimation heuristics which employ local data access observations, and demonstrate that the estimated data download times are fairly close to real measurements, with 90% of the estimates lying within 0.5 relative error in live experimentation on PlanetLab.
- We infer the latency to the server based on the prior neighbor measurement without explicitly probing the server. For this, we extend existing estimation heuristics [20, 17, 29] to more accurately work with a *limited* number of neighbors. Our enhancement gives accurate results even with only a few neighbors.
- We present accessibility-aware resource selection techniques based on our estimation functions and compare to the optimal and standard techniques such as proximity-based and random selection. Our results indicate that our approach not only outperforms the standard techniques, but does so over a wide range of operating conditions.
- We investigate the impact of churn prevalent in loosely-coupled distributed systems. The results show that our techniques perform satisfactorily even under a high degree of churn.

2 Accessibility-based Resource Selection

In this section, we first present our system model followed by an overview of the accessibility-based resource selection algorithm that uses data accessibility to select appropriate compute nodes in the system.

2.1 System Model

Our system model consists of a network of *compute nodes* that provide computational resources for executing application jobs, and *data nodes* that store data objects required for computation. In our context, data *objects* can be files, database records, or any other data representations. We assume both compute and data nodes are connected in an overlay structure. We do not assume any specific type of organization for the overlay. It can be constructed by using typical overlay network architectures such as unstructured [7] and structured [40, 37, 38, 45], or any other techniques. However, we assume that the

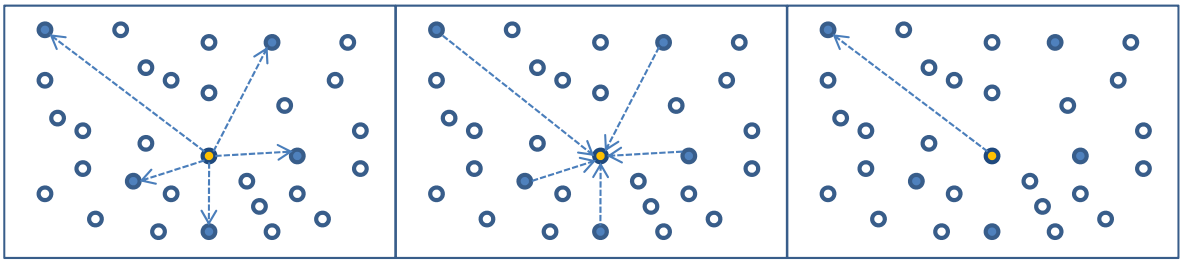


Figure 1. Accessibility-based resource selection

system provides built-in functions for object *store* and *retrieval* so that objects can be disseminated and accessed by any node across the system. Each node in the network can be a compute node, data node, or both.

Since scalability is one of our key requirements, we do not assume any centralized entities holding system-wide information. For this reason, any node in the system can submit a *job* in our system model. A job is defined as a unit of work which performs computation on an *object*. To allocate a job, a submission node, called an *initiator*, selects a compute node from a set of *candidates*. We assume the use of a resource discovery algorithm to determine the set of candidate nodes, and leave the efficient discovery of candidates as future work. Many existing resource discovery algorithms (e.g., SWORD [30]) can be used to return a set of matching candidate nodes, though they may not consider locality to data. Once the initiator selects a node, the job is transferred to the selected node, called a *worker*. The worker then downloads the data object required for the job from the network and performs the computation. When the job execution is finished, the worker returns the result to the initiator.

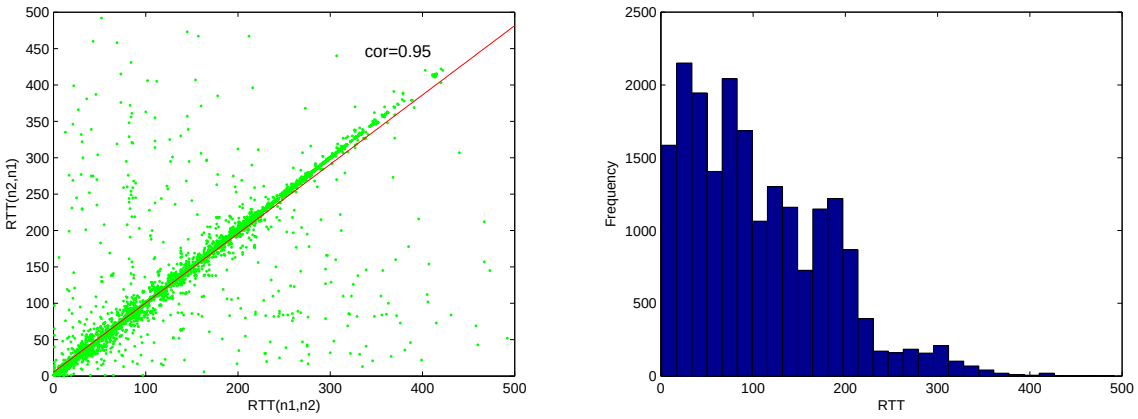
Formally, job J_i is defined as a computation unit which requires object o_i to complete the task. We assume that objects, e.g. o_i, have already been staged in the network and perhaps replicated to a set of nodes $R_i = \{r_{i1}, r_{i2}, ...\}$ based upon projected demand. The job J_i is submitted by the initiator. From the given candidates $C = \{c_1, c_2, ...\}$, the initiator selects one (i.e., $worker \in C$) to allocate the job.

2.2 Resource Selection

Figure 1 illustrates the resource selection process in our system model once the initiator has a set of candidate nodes to choose from. To select one of the given candidates, the initiator may simply make a choice randomly. However, relying on such randomness does not generally satisfy given system performance goals as we will show. Instead, querying the candidates will allow the system to acquire relevant information for job allocation, since there is no entity with global information (Figure 1(a)). The candidate offers the relevant information (Figure 1(b)). One type of information would be proximity to the server which is often used in today's distributed systems. Based on the information, the initiator makes a decision, and allocates the job to the selected worker (Figure 1(c)). To incorporate the impact of data access on the performance of job execution, our goal is to select the *best* candidate node in terms of accessibility to a data node (server) holding object o_i. Further, due to the decentralized nature of our system, we would like to make this selection without assuming any global knowledge.

To achieve this goal, we use an accessibility-based ranking function to rank the different candidate nodes. Since our goal is to maximize the efficiency of data access from the selected worker node, we use the expected *data download time* as the metric to quantify accessibility. Thus, given a set of candidates C for job J_i that requires access to object o_i, each candidate node c_m returns its accessibility $accessibility_{c_m}(J_i)$ in terms of the estimated download time for the object o_i, and the initiator then selects the node with the smallest accessibility value. Note that since we are assuming lack of any global knowledge, these estimates are based on the local information available to the individual candidate nodes. Therefore, sometimes it is possible that the candidate cannot provide any meaningful estimate of its accessibility to the required data object. In this case, the candidate simply returns value of *infinity* as its estimated download time for the object, indicating the lack of any information. The initiator would filter out such a candidate. If all candidates return *infinity*, one of the candidates is randomly selected. Formally, the selection heuristic H_s is defined as follows:

$$H_s : C \rightarrow c_m \text{ such that}$$



(a) RTT Symmetry

(b) RTT Histogram

Figure 2. Ping map statistics

$$accessibility_{c_m}(J_i) = \min_{n=1,..,|C|}(accessibility_{c_n}(J_i)).$$

Having described the accessibility-based resource selection algorithm, the question is how the candidate nodes can estimate their accessibility using local information (e.g., even without having downloaded object o_i in the past), and what factors they can use for this estimation. We explore this question in the next section.

3 Accessibility Estimation

3.1 Accessibility Parameters

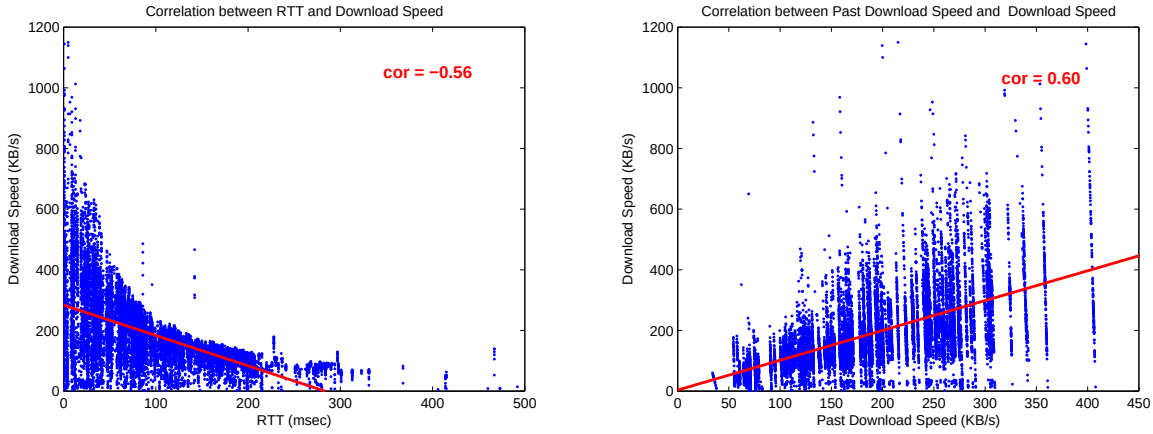
To answer the above question, we first investigate what parameters would impact accessibility in terms of data download time. Intuitively, a node's accessibility to a data object will depend on two main factors: the *location* of the data object with respect to the node, and the node's *network characteristics*, such as its connectivity, bandwidth, and other networking capabilities. We have explored a variety of parameters to characterize these factors, and found some interesting correlations. For this characterization, we conducted experiments on PlanetLab [33] with 133 hosts over three weeks. In these experiments, 18 2MB data objects were distributed over the nodes, and over 14,000 download operations were carried out to form a detailed trace of data download times. To measure inter-node latencies, an ICMP ping test was repeated nine times over the 3-week period, and the minimal latency was selected to represent the latency for each pair. We next give a brief description of the main results of this study.

We begin by introducing the ping statistics between nodes in the experimental environment. Figure 2 shows symmetry and distribution of round trip time (RTT). We can see strong symmetry between pairs in the figure — 94% of the total pairs have less than 5 ms difference. The distribution of latency is left-skewed and heavy-tailed, so the average (107 ms) is a bit larger than the median (87 ms).

The first result is the correlation of latency and download speed between node pairs. Intuitively, it is likely that smaller latency between client and server would lead to better performance in downloading. Figure 3(a) plots the relationship between RTT and download speed (defined as the ratio of downloaded data size and download time)[1]. We find a negative correlation $(r = -0.56)$ between them, indicating that *latency can be a useful factor when estimating accessibility between node pairs.*

In addition, we discovered a correlation between the download speed of a node for a given object and the past average download speed of the node, as shown in Figure 3(b) ($r = 0.6$). The intuition behind this correlation is that past download behavior may be helpful to characterize the node in terms of its network characteristics such as its connectivity and bandwidth.

[1]We use download speed to make our results independent of object size.



(a) Correlation of RTT and download speed

(b) Correlation of download speed and past average download speed

Figure 3. Correlations of access parameters

For example, if a node is connected to the network with a bad access link, it is almost certain that the node will yield low performance in data access to any data source. This result suggests that *past download behavior of a node can be a useful component for accessibility estimation*.

We had explored other parameters as well, but we could not find meaningful correlations. We introduce some of them here. We investigated the impact of latency affinity between two nodes. In this setting, for all combinations of three nodes, say a, b, and c, we compared $RTT(a,b)$ to $|RTT(a,c) - RTT(b,c)|$. We found a pretty strong correlation $(r = 0.66)$, which means that if $RTT(a,b)$ is small, the difference to the other one (i.e. $|RTT(a,c) - RTT(b,c)|$) is also small with strong correlation. However, when latency rises beyond some point (e.g. $RTT(a,b) > 250ms$), the results began to oscillate, suggesting little significance. This phenomenon can be also explained by the triangle inequality [14, 41]. Another example is a correlation of latency and download elapsed time between a pair. The basic question here is that if the RTT between two nodes is small, then two nodes would show similar behaviors in downloading. In other words, two nodes would have similar downloading capability if they resided on the same local or regional network. However we could not find any noticeable correlation in this experiment $(r = 0.1)$.

Based on the statistical correlations we discovered, we next present estimation techniques to predict data access capabilities of a node for a data object. Note that we will not assume global knowledge of these parameters (e.g., pairwise latencies between different nodes), but use hints based on local information at candidate nodes to get accessibility estimates. It is worth mentioning that it is not necessary to estimate the exact download time; rather our intention is to rank nodes based on accessibility so that we can choose a *good* node for job allocation. Nonetheless, if the estimation has little relevance to the real performance, then the ranking may deviate far from the desired choices. Hence we require that the estimation techniques demonstrate sufficiently accurate results which can be bounded within any tolerable error range.

3.2 Self-Estimation

As described above, latency to the server and download speed of a node are useful to assess its accessibility to a data object. We first provide an estimation technique that uses historical observations made by a node during its previous downloads to estimate these parameters. Note that these past downloads can be to any objects and need not be for the object in question. We refer to this technique as *self-estimation*.

To employ past observations in the estimation process, we assume that the node records access information it has observed. Suppose $\mathcal{H}_h^i$ is the i-th download entry at host h. This entry includes following information: object name, object size, download elapsed time, server, distance to server, and timestamp. As a convention, we use *dot*(.) notation to refer to an item of the entry, for example, $\mathcal{H}_h^i.size$ represents the object size in i-th observation at host h.

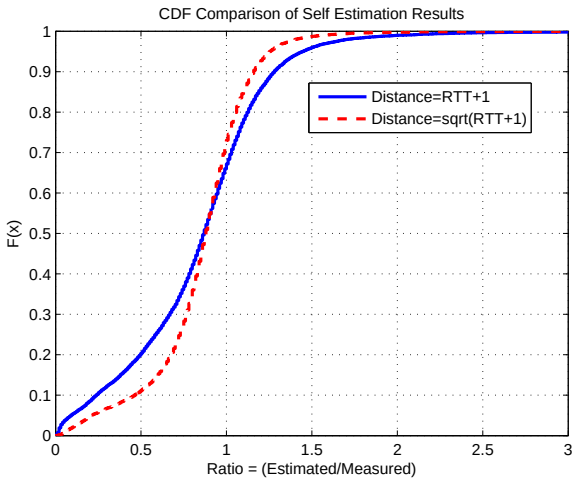


Figure 4. Self estimation result

We first estimate a *distance* factor between the node and the server, based on their inter-node latency. We consider two latency models for the distance metric: *RTT* and *square-root of RTT*. These are often used in TCP studies to cope with congestion efficiently to improve system throughput. Studies of window-based [32] and rate-based [31] congestion control revealed that RTT and square-root of RTT are inversely proportional to system throughput, respectively. We consider both latency models for the distance metric and compare them to see which is preferable later in this section. The mean distance of node h to the servers it has seen so far is then calculated as follows:

$$\overline{Distance_h} = \frac{1}{|\mathcal{H}_h|} \cdot \sum_{i=1}^{|\mathcal{H}_h|} \mathcal{H}_h^i.distance$$

We then characterize the network characteristics of the node by estimating its mean download speed based on prior observations:

$$\overline{DownSpeed_h} = \frac{1}{|\mathcal{H}_h|} \cdot \sum_{i=1}^{|\mathcal{H}_h|} \frac{\mathcal{H}_h^i.size}{\mathcal{H}_h^i.elapse}$$

Using the above factors, we estimate the expected download time for a host h to download object o as:

$$SelfEstim_h(o) = \delta \cdot \frac{size(o)}{\overline{DownSpeed_h}} \tag{1}$$

where

$$\delta = \frac{distance_h(server(o))}{\overline{Distance_h}}$$

Here, $size(o)$ means the size of object o, $server(o)$ means the server for object o, and $distance_a(b)$ means the distance between nodes a and b.

Intuitively, The parameter δ gives a ratio of the distance to the server for object o to the mean distance it has observed so far. Smaller δ means that the distance to the server is closer than the average distance the node has seen so far, and hence its estimated download time is likely to be smaller than previous downloads. The other part of Equation 1 uses the mean download speed to derive the estimated download time as being proportional to the object size.

Figure 4 shows the results of self-estimation. In the figure, the x-axis is ratio of the estimated time to the real measured time. Thus a ratio of 1 means that the estimation is exactly correct. As shown in the figure, we can see that $\sqrt{RTT}$ yields better estimation results than the simple RTT. Using $\sqrt{RTT}$ almost 90% of the total estimations occur within a ratio between 0.5 and 1.5 (i.e. relative error = 0.5). In contrast, the simple RTT yields 76% of the total estimations within the same error margin. Based on this result, we set $distance = \sqrt{RTT + 1}$, where RTT is expressed in milliseconds[2]. With this distance

[2]we add 1 to avoid division by zero.

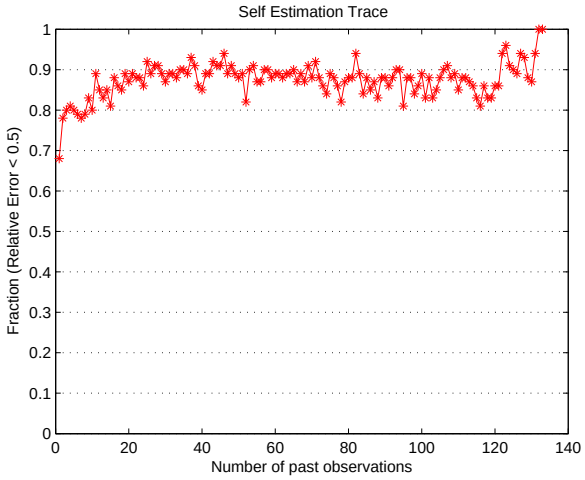


Figure 5. Self estimation trace result

metric, we can see that a significant portion of the estimations occur around the ratio, indicating that the estimation function is fairly accurate. We will see in Section 5 that this level of accuracy is sufficient for use as a ranking function to rank different candidate nodes for resource selection.

We also found that the self estimation produces accurate results with a limited number of observations. Figure 5 demonstrates how the estimation results improve with the cumulated observations over time. In the figure, the x-axis is the number of past observations, while the y-axis is the fraction of estimations which occur within a relative error of 0.5. Initially, the fraction is quite small below 0.7, but it sharply increases as observations have been accrued. With only 10 observations, for example, the fraction goes beyond 0.8, and approaches to 0.9 with 20 observations, suggesting that *the self estimation yields sufficiently accurate results to rank nodes with respect to accessibility with a small number (~ 10) of past observations.*

Since self-estimation is not required to have prior observations for the object in question, it must first search for the server and then determine the network distance to it. *Search* is often done by flooding in unstructured overlays [7], or by routing messages in structured overlays [40, 37, 38, 45], which may introduce extra traffic. Distance determination would require probing which adds additional overhead.

3.3 Neighbor Estimation

While self-estimation uses a node's prior observations to estimate the accessibility to a data object, it is possible that the node may have only a few prior download observations (e.g., if it has recently joined the network), which could adversely impact the accuracy of its estimation. Further, as mentioned above, self-estimation also needs to locate the object's server and determine its latency to the server to get a more accurate estimation. This server location and probing could add additional overhead and latency to the resource selection.

To avoid these problems, we now present an estimation approach that utilizes the prior download observations from a node's neighbors in the network overlay for its estimation. We call this approach *neighbor estimation*. The goal of this approach is to avoid any active server location or probing. Moreover, by utilizing the neighbors' information, it is likely to obtain a richer set of observations to be used for estimation. However, the primary challenge with using neighbor information is to correlate a neighbor's download experience to the node's experience given that the neighbor may be at a different location and may have different network characteristics from the node.

To assess the downloading similarity between a candidate node and a neighbor, we first define the notion of *download power (DP)* to quantify the data access capability of a node. The idea is that a node with higher DP is considered to be superior in downloading capability to a node with lower DP. We formulate DP for a host h as follows:

$$DP_h = \frac{1}{|\mathcal{H}_h|} \sum_{i=1}^{|\mathcal{H}_h|} \Big(\frac{\mathcal{H}_h^i.size}{\mathcal{H}_h^i.elapse} \times \mathcal{H}_h^i.distance \Big) \tag{2}$$

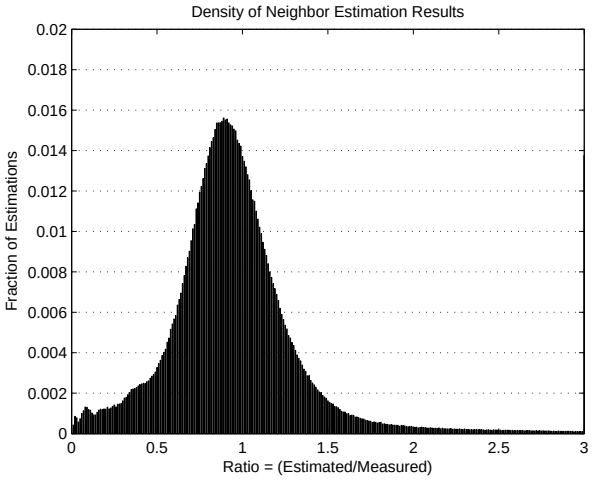


Figure 6. Neighbor estimation result

Intuitively, this metric combines the metrics of download speed and distance defined in the previous subsection. As seen from Equation 2, *DP* $\propto$ *download speed*, which is intuitive, as it captures how fast a node can download data in general. Further, we also have *DP* $\propto$ *distance to the server*, which implies that for the same download speed to a server, the download power of a node is considered higher if it is more distant from the server. Consider an example to understand this relation between download power and distance. Suppose that two client nodes, one in the US and one in Asia, access data from servers located in the US. Then, if the two clients show the same download time for the same object, the one in Asia might be considered to have better downloading capability for more distant servers, as the US client's download speed could be attributed to its locality. Hence, access over greater distance is given greater weight in this metric. To minimize the effect of download anomalies and inconsistencies, we compute DP as the average across its history of downloads from all servers.

Now, we define a function for neighbor estimation at host h by using information from neighbor n for object o:

$$NeighborEstim_h(n, o) = \alpha \cdot \beta \cdot elapse_n(o) \tag{3}$$

where

$$\alpha = \frac{DP_n}{DP_h}, \beta = \frac{distance_h(server(o))}{distance_n(server(o))},$$

and $elapse_n(o)$ is the download time observed by the neighbor for the object. It is possible that the neighbor has multiple observations for the same object, in which case we pick the smallest download time as the representative.

Intuitively, to estimate the download time for object o based on the information from neighbor n, this function first of all uses the relevant download time of the neighbor. As a rule, the estimation result is the same if all conditions are equivalent to the neighbor. To account for differences, we employ two parameters α and β. The parameter α compares the download powers of the node and the neighbor for similarity. If the DP of the node is higher than the neighbor, the function gives smaller estimation time because the node is considered superior to the neighbor in terms of accessibility. The parameter β compares the distances to the server, so that if the distance to the server is closer for the node than the neighbor's, the resulting estimation will be smaller[3].

Figure 6 illustrates the normalized histogram of neighbor estimation. The x-axis is the ratio of the estimation result to the real measured value, while the y-axis is the fraction of the estimations. As seen from the figure, a substantial portion of the estimated values are located near the ratio 1. Nearly 85% of estimations reside within a relative error 0.5. This suggests that *neighbor estimation produces useful hints to rank nodes with respect to accessibility.*

To realize neighbor estimation, it is necessary to gather information from the neighbors. As can be seen in the *NeighborEstim* function, it requires aggregated information from the neighbor: DP, distance to the server, and the download elapsed time for the object. This payload is quite small and has low network overhead.

[3]We discuss below how the server distance can be estimated without active probing.

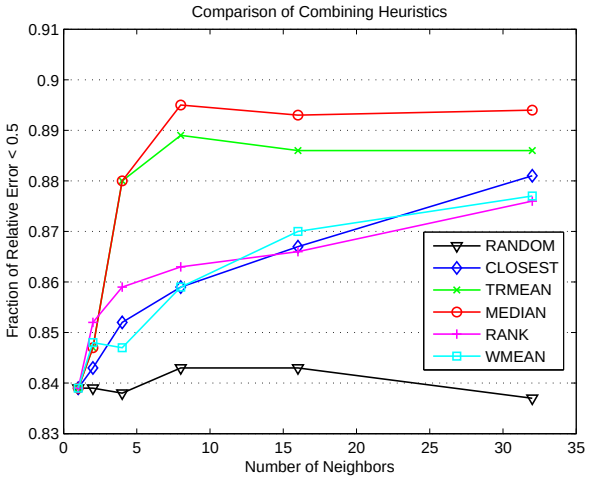


Figure 7. Comparison of combining heuristics

Although neighbor estimation provides a useful hint for assessment of accessibility, multiple neighbors can provide different information for the object. Hence the next question is *how can we efficiently combine information from different neighbors to obtain more accurate results*? We consider the following heuristics to combine the information from neighbors:

- RANDOM, which selects a random neighbor for estimation.
- CLOSEST, which selects the closest neighbor in terms of distance.
- TRMEAN, which takes trimmed mean of all estimations except both the lowest and the highest.
- MEDIAN, which takes the median value of all estimations
- RANK, which ranks the neighbors with a ranking function $rank(n_i) = \omega \cdot \frac{distance_c(n_i)}{MaxDistance} + (1-\omega) \cdot \frac{|DP_{n_i} - DP_c|}{DP_c}$ for $i = 1, .., |N|$, where c is the node, n_i is a neighbor, $|N|$ is the neighbor size, and $MaxDistance$ is a constant to define maximal distance (e.g. $MaxDistance = \sqrt{500}$). This heuristic gives higher rank to a neighbor which is closer to the server in distance and has similar DP with the node, and selects the estimation of the highest ranked neighbor. We set ω to 0.5.
- WMEAN, which gives a weight $weight(n_i) = 1 - rank(n_i)$ to neighbor n_i (i.e. $\sum_i weight(n_i) = 1$), and then the estimated results are weighted and averaged, thus resulting in one estimation value.

Figure 7 compares combining heuristics with respect to the number of neighbors. Overall, MEDIAN and TRMEAN work better than others. For both heuristics, the fraction reaches 0.88 with 4 neighbors, as shown in the figure. Given that the number of neighbors providing relevant observations may be limited in many cases, we believe that either MEDIAN or TRMEAN will be a good choice. For this reason, we use MEDIAN as the combining heuristic as follows:

$$NeighborEstim_h(o) = median_{n_i \in N}(NeighborEstim_h(n_i, o))$$

3.4 Inferring Server Latency without Active Probing

While the neighbor estimation requires latency to the server as a parameter (Equation 3), we can avoid the need for active probing by exploiting the server latency estimates obtained from the neighbors themselves. This makes it possible to minimize additional overhead in estimation in terms of server location and pinging.

According to the study in [41], a dominant portion of total paths ($\gg 90\%$) satisfied the property of triangle inequality. We also observed that 95% of total paths in our data satisfied this property. The triangulated heuristic estimates the network

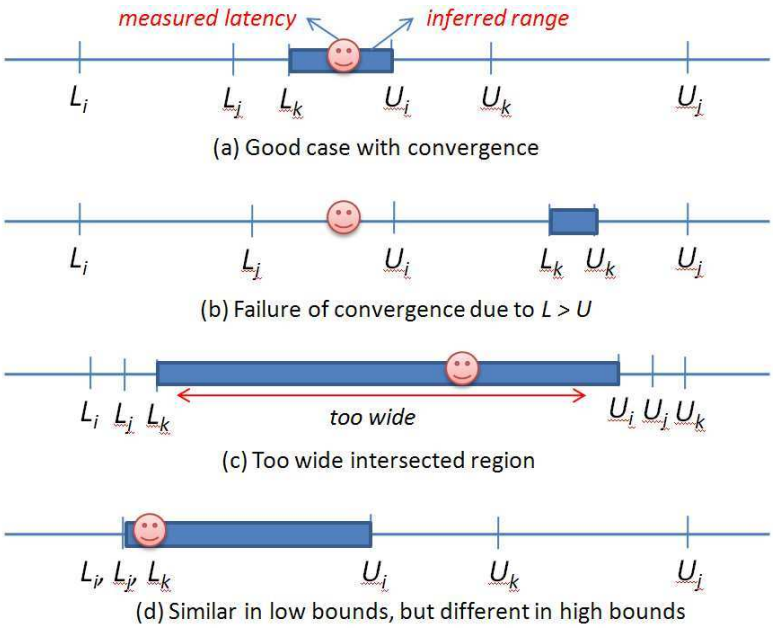


Figure 8. Example cases of inference

distance based on this property. It infers latency between peers with a set of landmarks which hold precalculated latency information between the peers and themselves [29]. The basic idea is that the latency of node a and c may lie between $|latency(a,b) - latency(b,c)|$ and $latency(a,b) + latency(b,c)$, where b is one of landmarks ($b \in B$). With a set of landmarks, it is possible to obtain a set of lower bounds (L_A) and upper bounds (U_A). If we define $L = \max(L_A)$ and $U = \min(U_A)$, then the range $[L, U]$ should be the tightest stretch with which all inferred results may agree. For the inferred value, Hotz [20] suggested L based on A*, Guyton and Schwartz [17] employed $(L + U)/2$, and most recently Eugene and Zhang reported U performs better than the others [29].

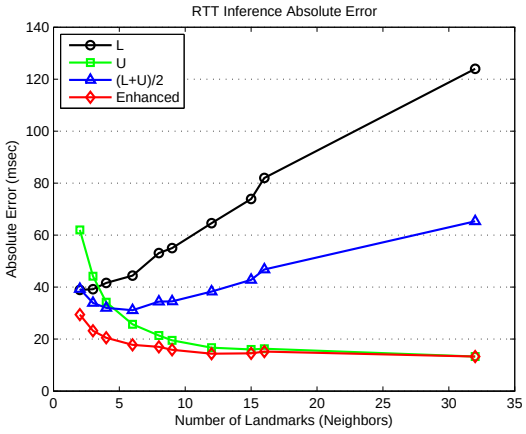
In our system model, we can use neighbors as the landmarks because they hold latency information both to the candidate and to the object server. By applying the triangulated heuristic, therefore, we can infer the latency between the candidate and the server without probing. However we found that the existing heuristics are inaccurate with a small number of neighbors which may be common in our system model. Hence we enhance the triangulated heuristic to account for a limited number of neighbors.

Our approach works by handling several situations that contribute to inaccuracy. Figure 8 shows example cases of inference based on the triangle inequality property. There are three landmarks which offer a pair of low and high bounds like (L_i, U_i), (L_j, U_j), and (L_k, U_k). Figure 8(a) shows a desired case in which the convergence successfully has been occurred.

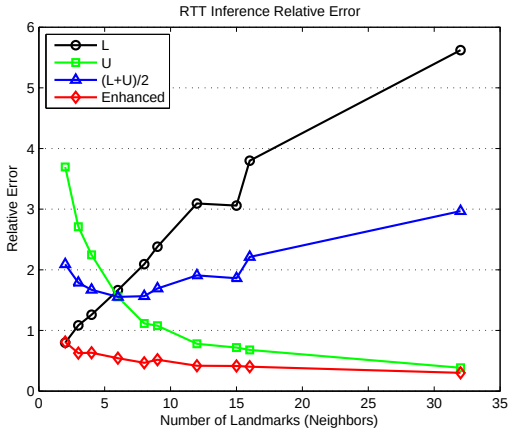
However, in some cases, it might fail to converge. As mentioned, a majority of the paths satisfy the property of triangle inequality, but a few paths do not. We observed some cases of $L > H$ for which triangle inequality does not hold. Consider the following situation: all but one landmark give reasonable latencies, but if that one gives fairly large low and high bounds (e.g. because it is connected with a relatively slow link) , the expected convergence would not occur, thus leading to an inaccurate answer. Figure 8(b) illustrates this problem. In this case, picking any one of the existing approaches is erroneous. To overcome this problem, we remove all $L_i \in L_A$ which is greater than U, so we can make a new stretch that satisfies $L < U$. After doing so, we observed that taking simple mean produces greatly better results than the existing approaches.

Another potential problem takes place when all the landmarks are similarly apart from both nodes (e.g. a and c in the above explanation). Figure 8(c) illustrates a possible situation in which the stretch $[L, U]$ is too wide, so picking any one of L, U, and $(L + U)/2$ is likely to be highly inaccurate. Since it is an intrinsic limitation of the triangulated heuristic, we take the inaccurate result and leave it as a further study.

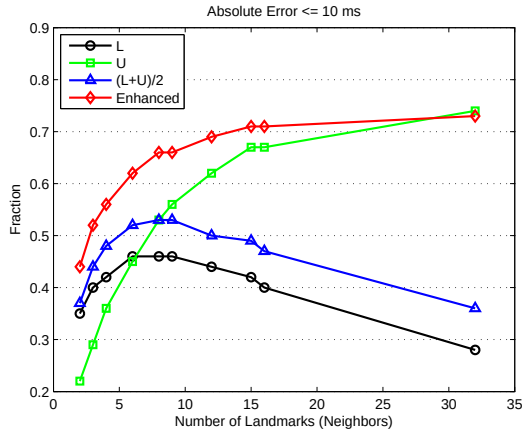
We also observed a problematic situation as illustrated in Figure 8(d). In this case, a significant portion of the inferred low bounds suggest similar values, but high bounds have a certain degree of variance. This usually happens where node c is close to a but the landmarks are all apart from node a. For this, we consider a weighted mean which is based on standard



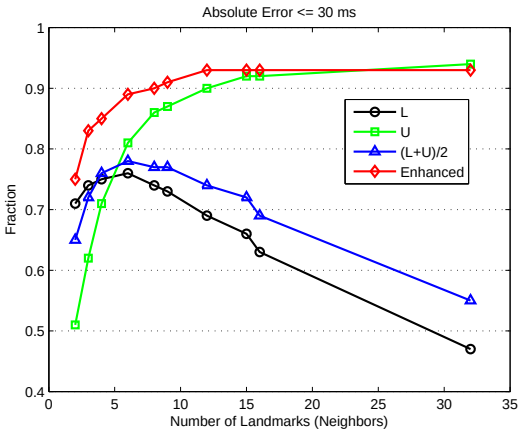
(a) Absolute Error



(b) Relative Error



(c) Fraction of Absolute Error $\leq$ 10 ms



(d) Fraction of Absolute Error $\leq$ 30 ms

Figure 9. Latency inference results

deviations (σ) of both low and high bounds:

$$mean = L \cdot \left(1 - \frac{\sigma_{L_A}}{(\sigma_{L_A} + \sigma_{U_A})}\right) + U \cdot \left(1 - \frac{\sigma_{U_A}}{(\sigma_{L_A} + \sigma_{U_A})}\right)$$

The intuition behind this is that if multiple inferred bounds suggest similar values for either low or high bounds, it is likely that the real latency is around that point. We employ this weighted mean if the heuristic fails to converge due to the stretch being too wide (e.g. $|L - U| > 50$, in our evaluation). If the heuristic can properly converge, we simply takes the simple mean (i.e. $(L + U)/2$).

We evaluated this enhanced heuristic with our data set. Even though our data set contains a relatively small number of nodes, we observed similar results with those reported in [29]. According to the results in [29], 90 percentile relative errors for the triangle heuristic taking U as the inferred value were 1.05 (6 landmarks), 0.8 (9 landmarks), 0.69 (12 landmarks), and 0.59 (15 landmarks). Our results show 1.26, 0.84, 0.62 and 0.57, respectively.

We report the evaluation results with the *absolute* error as well as the *relative* error for clarity. For example, if we think of two measured latencies 1 ms and 100 ms and the corresponding estimations 2 ms and 200 ms, then those two estimations give the same picture with respect to the relative error (i.e. relative error = 1, in this example). In contrast, they convey different

information with respect to absolute error. In fact, 1 ms difference is usually acceptable, but 100 ms error is not for latency inference.

Figure 9 demonstrates the inference results. As reported in [29], the heuristic employing U is overall better than the other two existing heuristics. However, we can see that our enhanced heuristic substantially outperforms the existing heuristics. In particular, the enhanced heuristic works well even when the number of landmarks is small. Since the number of neighbors which can offer the relevant latency information may be limited, the enhanced heuristic is desirable in our design. With 4 neighbors, the enhanced heuristic approximates 56% of the total estimations within 10 ms as shown in Figure 9(c), and 85% of the estimations are correctly estimated within 30 ms as shown in Figure 9(d).

A few words about the tolerance of error in the inferred result. Our estimation functions are not very sensitive to inferred results because the square-root of RTT is used. Thus, some range of errors are tolerable in our design. While those are tolerable, our focus is to exploit the neighbors' information. Other sophisticated latency prediction techniques can be used if available. To summarize, *the proposed technique estimates latency accurately without depending on explicit probing or other assumptions*.

4 Resource Selection Techniques

Resource selection is the process by which a node is chosen to execute the job. In this section, we present selection techniques based on the self and neighbor estimation techniques discussed in the previous section. Before doing this, we discuss random and proximity-based selections as standard techniques.

4.1 Standard Techniques

We consider *random* selection (RANDOM) and *proximity*-based selection (PROXIM) as standard techniques. The primary advantage of random selection is its simplicity. It simply selects a candidate at random. The price paid for this simplicity, is poor performance.

Proximity-based selection is a common choice in today's distributed systems. In this technique, a node with the smallest latency to the server is chosen. This is reasonable since we observed a significant correlation between RTT and download speed. To make it attainable, the relevant server must be discovered first, and then latency to the server from the candidates measured to identify which candidate node is closest to the server. In general, the process of server discovery and latency measurement introduces extra traffic.

4.2 SELF: Self Estimation-based Selection

In *self estimation-based selection* the accessibility is directly obtained from the self-estimation function:

$$accessibility_{c_m}(J_i) = \begin{cases} SelfEstim_{c_m}(o_i) & \text{if available} \\ infinite & \text{otherwise} \end{cases}$$

Note that the candidate need not have prior download experience with this object. The candidate provides the result of the self estimation function as its accessibility. However, the candidate returns *infinite* in the case that it cannot make any relevant estimation due to a lack of observations. Since the selection heuristic chooses the smallest accessibility, this candidate will be filtered out. However, if all candidates return *infinite*, a random selection is made[4]. Initially when the system starts, random selections will be made until the nodes have acquired download information.

The procedure is similar to one of the proximity-based selection technique, since this technique also requires server discovery and latency estimation if no prior interactions are recorded. This technique, therefore, may introduce additional traffic as well. The primary difference is in the selection principle. While the proximity-based technique considers only proximity for selection, this technique exploits the past access behaviors as well as proximity. For example, even if a node is fairly close to the server, it would not be chosen if it has shown bad performance in data access previously.

[4]In this case, it is possible to select a candidate by latency information as the proximity-based selection does; however, we do not consider such a fine-level optimization for simplicity.

Table 1. Download Traces

Trace	# of nodes	# of objects	# of downloads
1M	153	72	22,509
2M	231	83	25,934
4M	167	107	28,439
8M	158	85	26,105

4.3 NEIGHBOR: Neighbor Estimation-based Selection

Neighbor estimation-based selection employs neighbor estimation, so the candidate accessibility is derived from the neighbor estimation function:

$$accessibility_{c_m}(J_i) = \begin{cases} NeighborEstim_{c_m}(o_i) & \text{if available} \\ infinite & \text{otherwise} \end{cases}$$

Like self estimation-based selection, the candidate returns either the relevant estimation result or *infinite*. Then the selection heuristic will select the smallest estimation. If no candidate provides the relevant estimation, it simply performs random selection. The main difference from the self estimation-based selection is that this technique can reduce additional efforts for server discovery and latency measurement by sharing related information with neighbors. Thus *this technique should be useful more in environments where messaging cost is expensive so minimizing explicit interaction is crucial.*

4.4 HYBRID: Hybrid Selection Technique

One of the strong points of the neighbor estimation-based selection is the minimal messaging requirement. It collects relevant information directly from the neighbors. Such communication is local as opposed to querying the data servers. That is, it does not require additional traffic to search for objects or to measure distance to servers, if the relevant information exists. This is a significant benefit given the bandwidth scarcity in loosely-coupled distributed systems. On the other hand, this technique can make a poor choice if no neighbors have relevant observations.

In contrast, the self estimation-based selection may not diminish explicit probing. However, it is highly likely that some candidates can compute a self estimation because it does *not* require direct observation of the object. This leads to better results by avoiding random selection.

The hybrid selection technique incorporates these heuristics to exploit their benefits. In this technique, candidate nodes consult their neighbors first to acquire the relevant access information. If this information exists, they run neighbor estimation-based selection. If not, they attempt to compute accessibility using self estimation instead of sending an *infinite* value. By doing so, it is possible to diminish the chance of random selection which may degrade system performance. Therefore, the *hybrid selection technique would be helpful as it takes advantage of both the neighbor and self estimation-based selections*.

5 Evaluation

In this section, we evaluate our accessibility-based resource selection techniques. We employ two performance metrics for our evaluation: (1) *mean download time* and (2) *mean download time ratio* to optimal selection. The former is useful to understand the absolute performance, while the latter helps to compare the performance of different algorithms, complementing the former.

5.1 Experimental Setup

We conducted over 100K actual downloading for a span of 5 months with 241 PlanetLab nodes geographically distributed across the globe. For this, we deployed a Pastry [38, 15] network, a structured overlay based on a DHT ring. We distributed data objects of four sizes: 1M, 2M, 4M, and 8M bytes, over the network, each object with a unique key. Table 2 shows the distribution of objects. We then generated a series of random queries so that the selected nodes perform downloading the relevant objects. Table 1 provides the details of the download traces. In the simulations, we use a *mixture* of all traces rather than individual traces, unless otherwise mentioned.

Table 2. Object Distribution

# of objects	Fraction of nodes
0	0.29
1	0.26
2	0.25
3	0.10
4	0.10

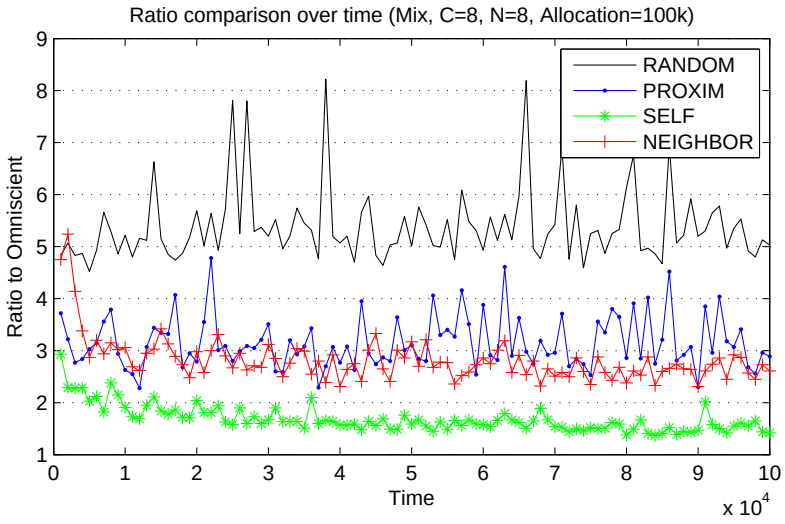


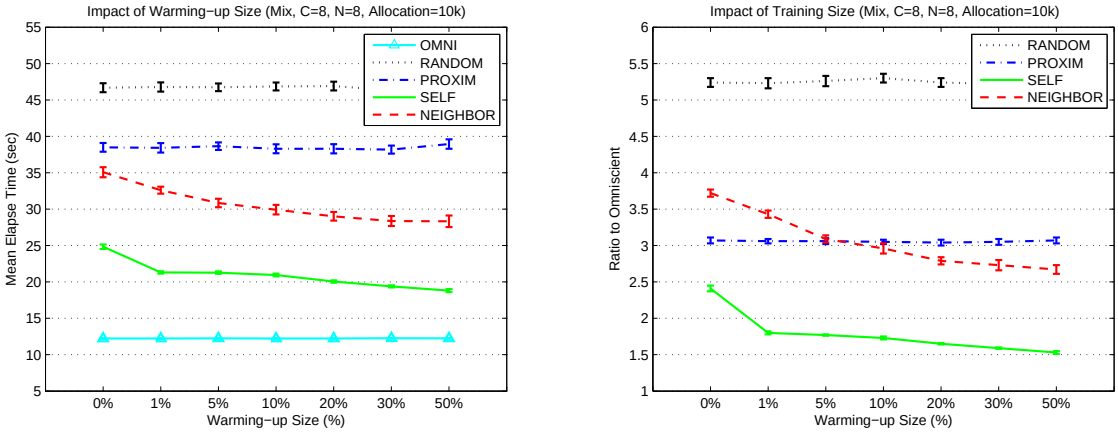
Figure 10. Performance over the time

To evaluate resource selection techniques, we design and implement a simulator which inputs the ping maps and the collective downloading traces and outputs performance results according to the selection algorithms. Initially, the simulator constructs a network in which nodes are connected to each other with a predefined neighbor size To minimize error due to the construction, we repeated simulations 50 times and report the results with 95% confidence intervals as needed. After constructing the network, the simulator runs each resource selection algorithm. Initially, it constructs a *virtual trace* in which the list of candidates and the download time from each candidate are recorded. The candidate nodes are randomly chosen for each allocation. As the candidate may have more than one actual download record for a server, the download time is also randomly selected from them. The simulator then selects a worker based on each selection algorithm. Based on the selected worker, the download time is returned from the virtual trace. One virtual trace is consumed in one simulation time step.

For our evaluation, we compare the resource selection techniques described in the previous section. In addition, we consider *omniscient* selection (OMNI) for the purpose of comparison which always returns the best node based on future knowledge of download times.

5.2 Performance Comparison over Time

We begin by presenting the performance comparison over time. Figure 10 compares the performance over the 100K consecutive job allocations. As the default, we set both the candidate size and the neighbor size to 8 (and it is applied to all the following experiments, unless otherwise mentioned). Overall the proposed techniques yield good results: SELF is the best across time and NEIGHBOR works better than PROXIM most of the time. RANDOM yields poor performance with a significant degree of variation, as expected. PROXIM is about 3 times of optimal with a relatively high degree of variation compared to the suggested techniques. NEIGHBOR is poor at first (due to warming up), but shows better results compared to PROXIM after about 6K simulation time steps. SELF works best approaching about 1.4 of optimal at the end of the simulation.



(a) Mean download elapsed time

(b) Mean ratio to optimal

Figure 11. Impact of warming-up size

5.3 Impact of Warming-up Size

As we observed that SELF approaches to the optimal over the time, we further investigate the impact of the warming-up size. We preloaded a portion of trace data in advance and observed the changes in performance. The preloading has been done with randomly selected observations in the trace. It it certain that RANDOM and PROXIM will not change since they do not employ past observations. In contrast, we expect that increasing warming-up size would benefit SELF and NEIGHBOR.

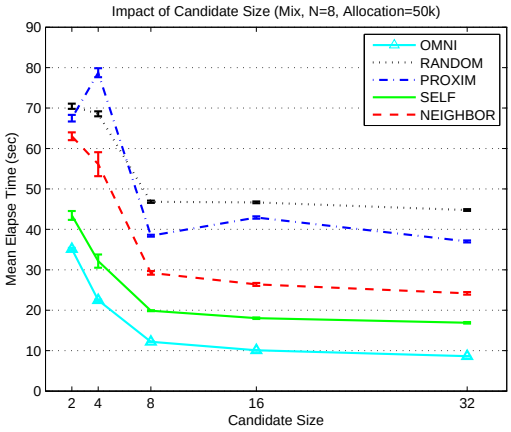
Figure 11 compares the performance of the selection techniques with respect to warming-up size. In this experiment, we attempted 10K trials to see the impact at the first stage. As expected, SELF approaches to optimal as warming-up size is greater, while RANDOM and PROXIM show little correlation. NEIGHBOR also improves as dramatic as SELF across warming-up size. It is interesting that NEIGHBOR always outperforms PROXIM in terms of mean download time as shown in Figure 11(a), while it shows a crossover between PROXM and NEIGHBOR in term of mean ratio to optimal in Figure 11(b). This is because that random selection in NEIGHBOR (due to lack of observations) contributes to the large ratio to optimal (see that average ratio of RANDOM is quite bigger than the others).

5.4 Impact of Candidate Size

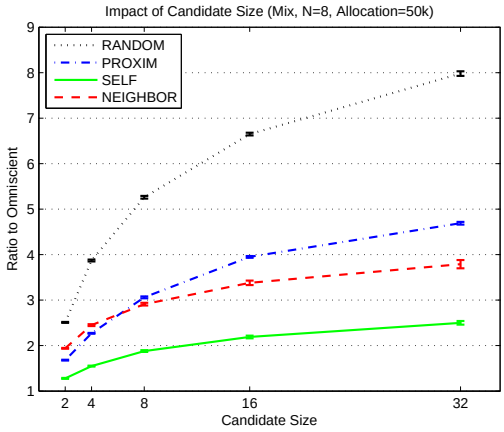
In our system model, a set of candidate nodes are evaluated for their accessibility before allocating a job. We now investigate the impact of candidate size ($|C|$). Figure 12 demonstrates the performance changes with respect to candidate size. In Figure 12(a), SELF continues to produce diminished elapsed times as the candidate size increases, yielding the best results among selection techniques. NEIGHBOR follows SELF with considerable gaps against the standard techniques. In particular, the proposed heuristics dramatically improve performance with $|C| = 8$, and then the improvement gain reduces after $|C| > 8$. Interestingly, PROXIM shows unstable results with greater fluctuation than RANDOM over the candidate sizes. In Figure 12(b), mean ratio to optimal increases along the candidate size. This is because OMNI has many more candidates to choose from, resulting in the larger performance gaps. However, we can see that the suggested techniques work better with more many candidates, making the slopes gentle compared to the standard ones. This result indicates that *the proposed techniques not only work better than standard ones across candidate sizes, but also further improve as the candidate size increases*.

5.5 Impact of Neighbor Size

We next investigate the impact of neighbor size on NEIGHBOR (the other heuristics are not affected by this parameter). Figure 13 shows how the selection techniques respond across the number of neighbors ($|N|$). As can be seen in the both

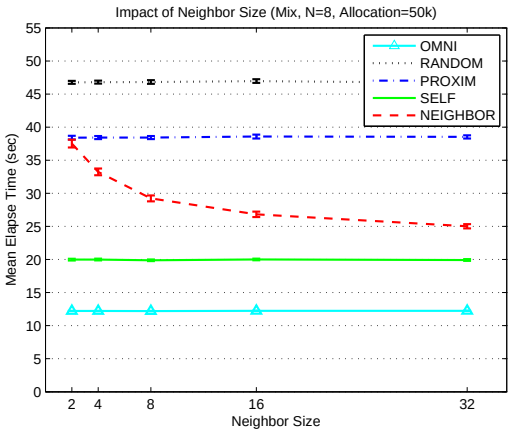


(a) Mean download elapsed time

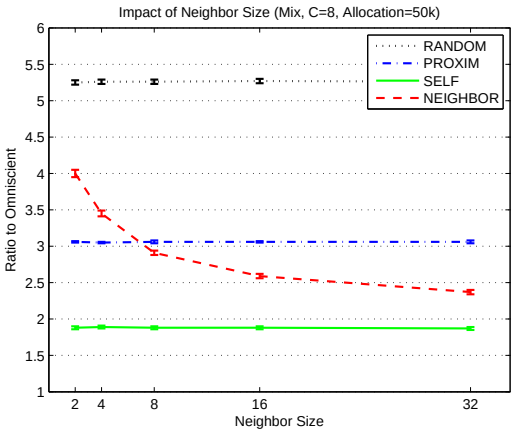


(b) Mean ratio to optimal

Figure 12. Impact of candidate size

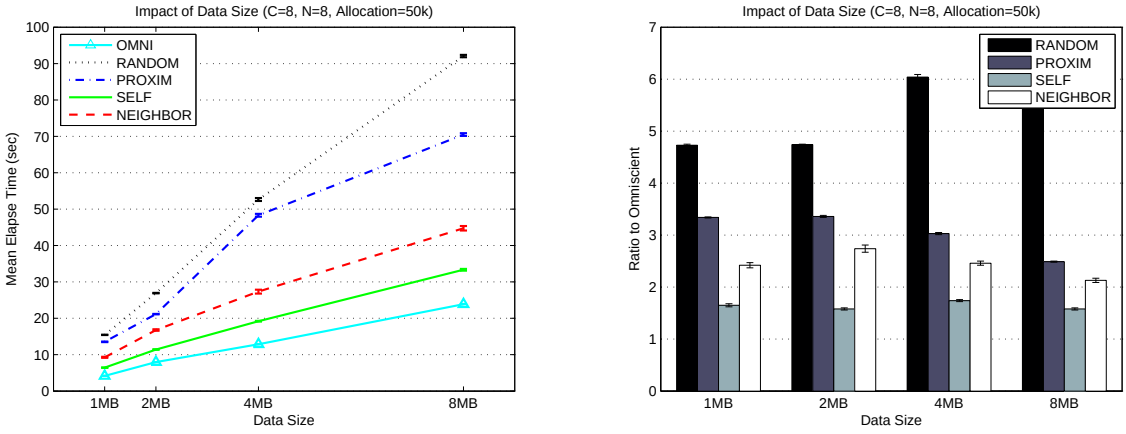


(a) Mean download elapsed time



(b) Mean ratio to optimal

Figure 13. Impact of neighbor size



(a) Mean download elapsed time

(b) Mean ratio to optimal

Figure 14. Impact of data size

performance figures, increasing the neighbor size dramatically improves the performance, while the others make no changes as expected. For example, the average download time in $|N| = 16$ is dropped to about 70% of the time for $|N| = 2$. The ratio to optimal is also dropped from 4.0 at $|N| = 2$ to 2.6 at $|N| = 16$. This is because it has more chances to obtain relevant observations with more many neighbors, thus decreasing the possibility of random selection. This result suggests that *NEIGHBOR will work better in environments where the node has connectivity with a greater number of neighbors*.

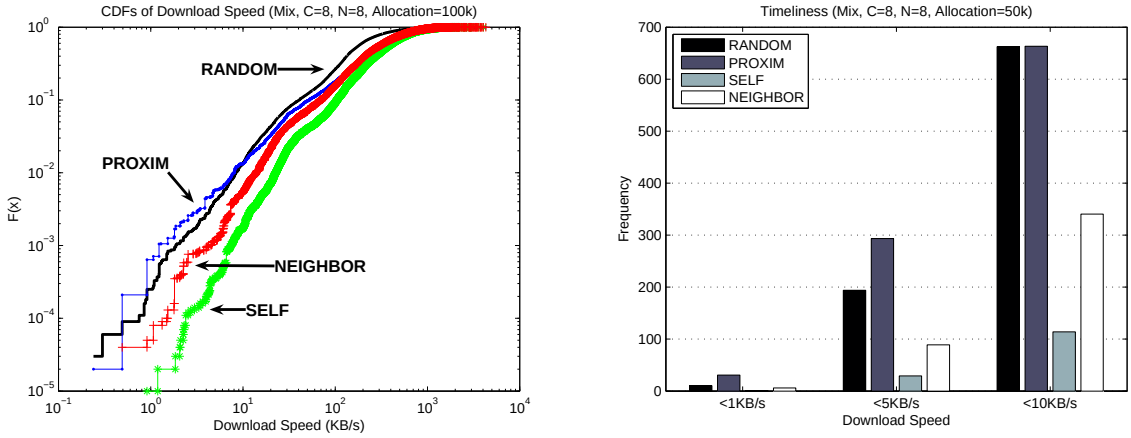
5.6 Impact of Data Size

We continue to investigate how the selection techniques work over different data sizes. Since the size of accessed objects can vary depending on applications in reality, selection techniques should work consistently across a range of data sizes. In this experiment, we run the simulation with *individual* traces rather than the mixture of the traces. Figure 14 compares performance with various data sizes. In Figure 14(a), we can see linear relationship between data size and mean download time. However, each technique shows a different degree of slope: SELF and NEIGHBOR increase more gently than the standard heuristics. With simple calculation, the slopes (i.e. $\Delta y/\Delta x$) of the techniques are RANDOM=10.9, PROXIM=8.1, SELF=3.8, and NEIGHBOR=5.1. In addition, Figure 14(b) shows that the suggested techniques consistently outperform the standard ones in terms of ratio to optimal regardless of data size. This result implies that *the proposed techniques not only work consistently across different data sizes, but they are also much more useful for data-intensive applications*.

5.7 Timeliness

It is crucial to choose *good* nodes for job allocation. On the other extreme, it is also important to avoid *bad* nodes when making a decision. For instance, selecting intolerably slow connections may lead to job incompletion due to excessive downloading cost or time-outs. However, it is almost impossible to pick good nodes every time because there are many contributing factors.

We observed how many times the techniques choose slow connections. Figure 15 shows this with two figures: cumulative distributions of the speed of connections with log-log scales and frequencies of the selection of bad connections. In both figures, we can see that the proposed techniques more often avoid slow connections. SELF most successfully excludes low speed connections, and NEIGHBOR also performs better than the standard techniques. When we count the number of poor connections selected, SELF has chosen under 5 KB/s connections less than 30 times, while PROXIM made over 290 selections which is almost an order of magnitude larger than SELF. One interesting result is that PROXIM selects poor connections more frequently than RANDOM (293 and 194 times respectively). This implies that *relying only on proximity information alone greatly increases the chance of very poor connections, thus leading to unpredictable response*



(a) CDF of download speed

(b) Frequency of low download speed

Figure 15. Timeliness

time. Compared to this, our proposed techniques successfully reduce chances to choose low speed connections by taking accessibility into account.

5.8 Impact of Churn

Churn is prevalent in loosely-coupled distributed systems. To see the impact of churn, we assume that mean session lengths of nodes are exponentially distributed. In this context, the session length is equivalent to the simulation time. For example, if the session length of a node is 100, the node changes its status to inactive after 100 simulation time steps. The node then joins again after another 100 time steps. We assume that nodes *lose* all past observations when they change status. Therefore, churn will have a greater impact on our selection techniques because we rely on historic observations. In contrast, the standard techniques suffer little from churn since they do not have any dependence on past observations. The virtual trace excludes objects for which the relevant servers are inactive.

We tested three mean session lengths: $s = 100$, $s = 1000$, and $s = 10000$, corresponding to *extreme*, *severe*, and *light* churn rates respectively. As can be seen in Figure 16, with both $s = 100$ and $s = 1000$, the number of active nodes dropped to around 50% right at the beginning of the simulation (≤ 1200 allocations), while it takes 10K simulation time steps to drop to 50% of inactive nodes with $s = 10000$. The reason why 50% of nodes are active (or inactive) is that departing nodes join again with the same probability of leaving. Hence, ultimately the fraction of active nodes is stabilized at 50% of total nodes throughout the simulation.

Figure 17 illustrates performance changes under churn. As mentioned, there is little impact on standard techniques. In contrast, our techniques are degraded in performance due to loss of observations. In Figure 17(a), SELF is comparable to PROXIM even under extreme churn. NEIGHBOR degrades and becomes worse than PROXIM under severe churn ($s = 1000$). This is because NEIGHBOR is likely to fail to collect the relevant observations, thus relying more on random selection, while SELF can perform reasonably accurate estimation with only a dozen of observations, as we have seen in Figure 5. Nonetheless, NEIGHBOR still works better than PROXIM in light churn ($s = 10000$) with lower overhead. In terms of ratio to optimal as shown in Figure 17(b), SELF is comparable to PROXIM under severe churn, but worse under extreme churn. NEIGHBOR also shows worse than PROXIM under light churn due to an increasing number of random selection attempts.

We now explore why NEIGHBOR suffers under severe and extreme churn. Figure 18 shows how NEIGHBOR performs under churn. The neighbor estimation rate means the fraction that NEIGHBOR successfully estimates based on the neighbor estimation, while the number of offered neighbors means that the average number of neighbors that provide observations when NEIGHBOR performs successfully, not relying on random selection. Under normal conditions, the neighbor estimation rate is almost 100% as soon as the simulation starts (Figure 18(a)), and the number of offered neighbors continuously

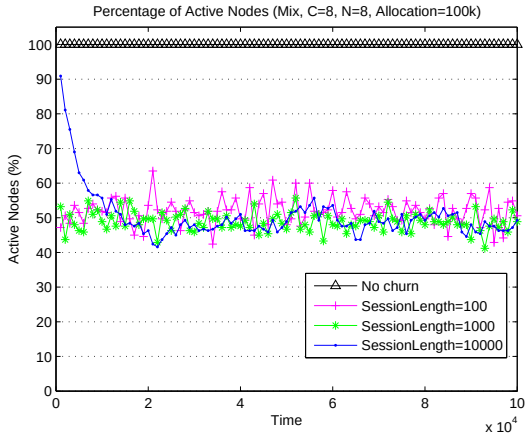
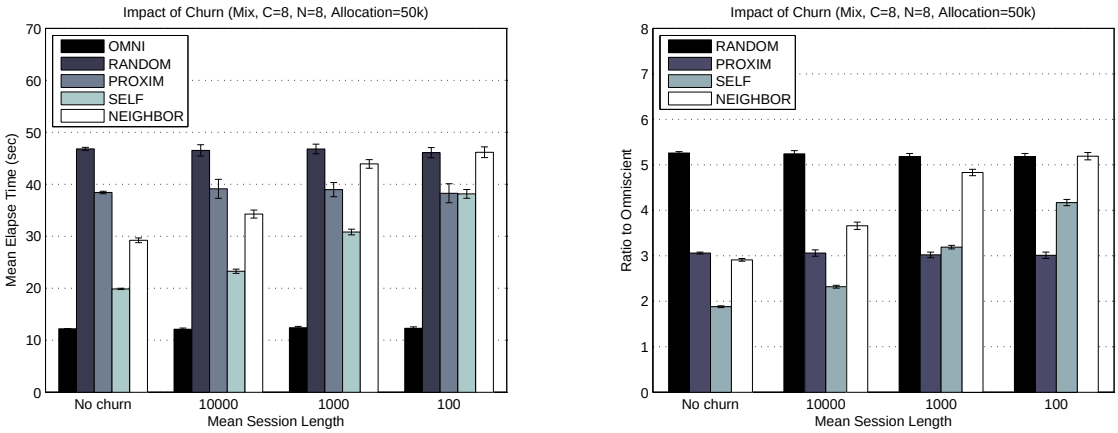


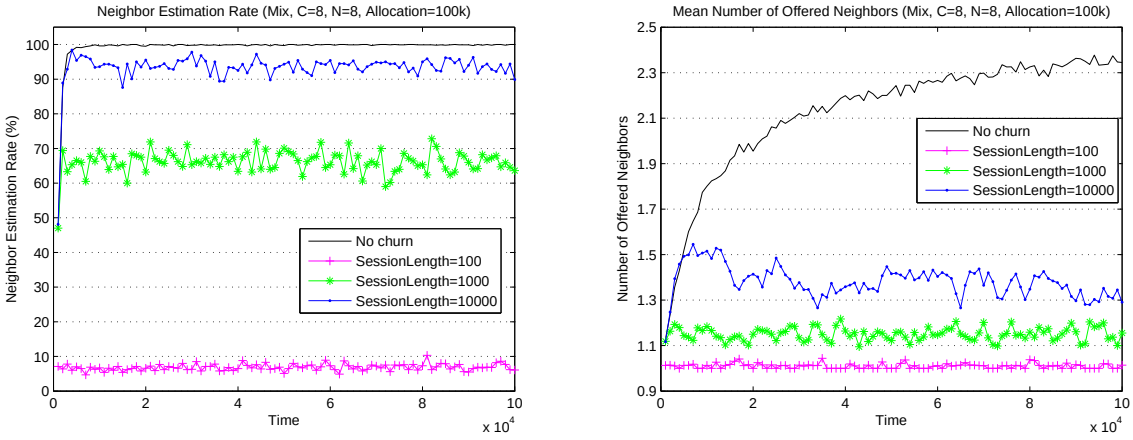
Figure 16. Active node ratio under churn circumstances



(a) Mean download elapsed time

(b) Mean ratio to optimal

Figure 17. Impact of churn



(a) Neighbor estimation rate

(b) Number of offered neighbors

Figure 18. Impact of churn in NEIGHBOR

increases over the time (Figure 18(b)). Under churn, both the neighbor estimation rate and the number of offered neighbors drop with respect to the degree of churn. Under light churn, the neighbor estimation rate is still over 90%, and the number of offered neighbors is 1.3–1.5 on average. Under severe churn, the estimation rate drops to 60–70%, implying that 30–40% of the decisions have been made by random selection. Under extreme churn, the neighbor estimation rate drops below 10%, so it essentially reduces to RANDOM.

To summarize, *the proposed techniques are stable under churn in which nodes suffer from loss of observations*. The result shows that SELF is at least comparable to PROXIM even under extreme churn, while NEIGHBOR is comparable to PROXIM when churn is light.

5.9 Performance of HYBRID

HYBRID combines the behavior of SELF and NEIGHBOR. Under normal conditions or light churn, it works like NEIGHBOR throttling extra traffic, but it encourages self estimation if the candidate can acquire no relevant information from the neighbors. By doing so, it diminishes the possibility of random selection.

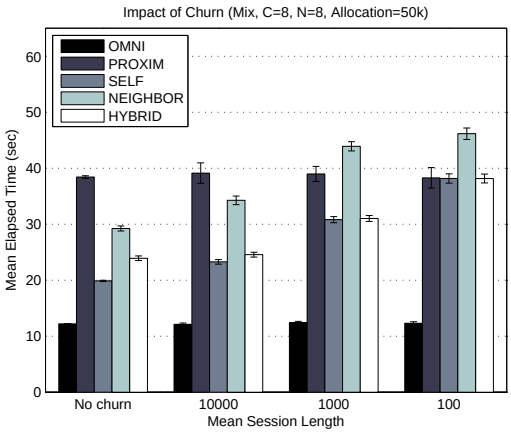
Figure 19 compares the performance of HYBRID with the other techniques. Under normal circumstance, HYBRID is a little worse than SELF, but it shows comparable results to SELF under churn. In all cases, HYBRID yields better results than NEIGHBOR. Figure 20 shows the self estimation rate under normal and churn circumstances. As can be seen in the figure, even under normal conditions, half of the candidate nodes perform self estimation as the basis for forming the accessibility value. Under extreme churn, most nodes rely on self observations, while 80–90% of nodes attempt self estimation under severe churn. Thus it can reduce the chance of random selection.

5.10 Impact of Replication

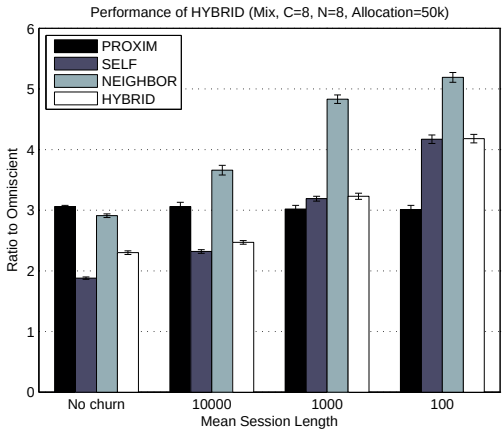
In loosely-coupled distributed systems, *replication* is often used to disseminate objects to provide *locality* in data access as well as high *availability*. We investigate the impact of replication to see if the proposed techniques consistently work in replicated environments.

For this, we construct replicated environments in which same-sized objects in the traces are grouped according to the replication factor and the object in the group is considered as a replica. The virtual trace is then constructed based on the group of the objects. In details, for all objects in the group, a randomly selected download time from each candidate is recorded in the virtual trace. The simulator then returns the download time according to the selected candidate and the server.

RANDOM will work same as in no replication environment with a random function to choose both a candidate and a replica server. PROXIM measures latencies from every candidate to every server, and then the pair with the smallest latency



(a) Mean download elapsed time



(b) Mean ratio to optimal

Figure 19. Performance of HYBRID

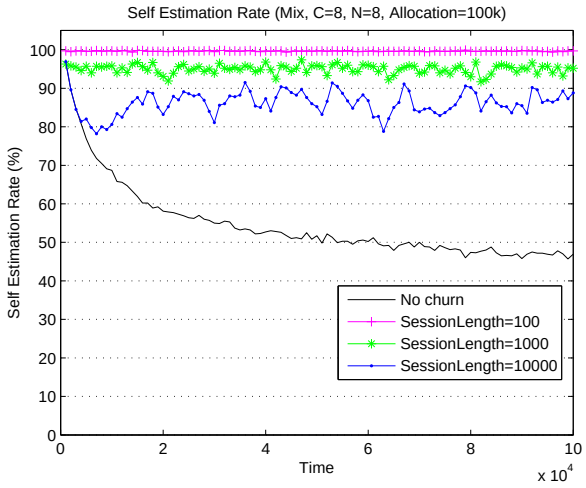
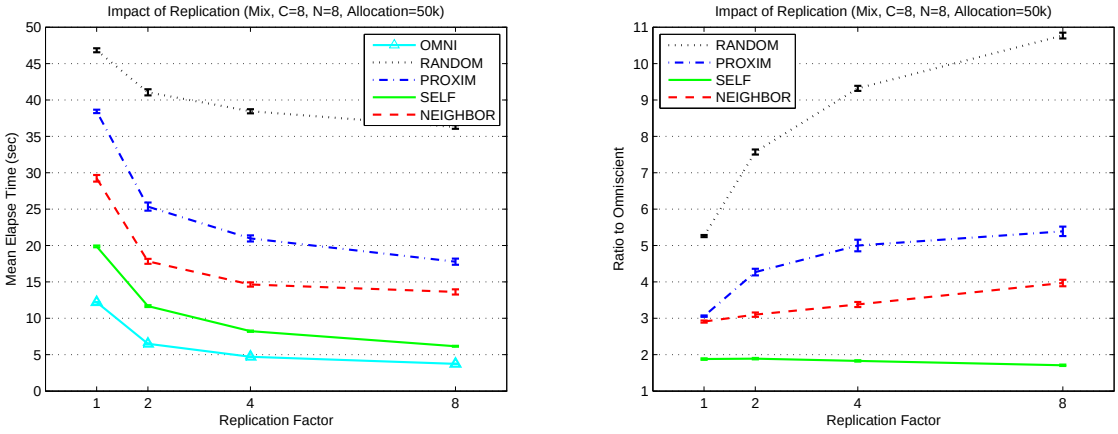


Figure 20. Self-estimation rate



(a) Mean download elapsed time

(b) Mean ratio to optimal

Figure 21. Performance under replicated environments

will be selected. SELF is similar to PROXIM: each candidate calculates the accessibility for each server, and reports the best one. In the case of NEIGHBOR, the candidate gathers all the relevant information from the neighbors. If it finds more than one server, $NeighborEstim(o)$ function is performed against each server, and then the best one is reported to the initiator. For both SELF and NEIGHBOR, the initiator finally selects the candidate with the best accessibility.

Figure 21 shows performance changes across replication factors (R_f). It is likely that the performance of all selection techniques improve as the replication factor increases because of data locality, and the result agrees with this expectation, as shown in Figure 21(a). PROXIM has significantly diminished mean download time (nearly half) under replication, but it is still behind the proposed techniques. SELF and NEIGHBOR outperform the standard techniques over all the replication factors. In Figure 21(b), we can see that SELF further reduces ratio to optimal as replication factor increases, while the others increase. In the meantime, NEIGHBOR widen gaps against standard techniques along replication factors. For example, the gap between NEIGHBOR and PROXIM is greatly widened at $R_f = 2$ compared to $R_f = 1$ (i.e. no replication).

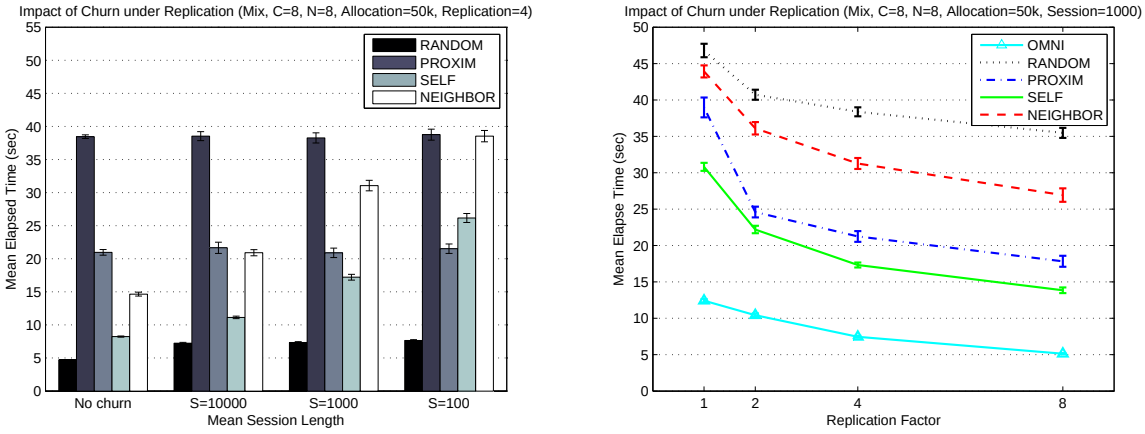
Figure 22 demonstrates the impact of churn under replicated environments. In Figure 22(a), we fix the replication factor at 4, and observed the performance change over a set of mean session lengths. By and large, the results are similar with the ones in the non-replicated environment. However, SELF is a little worse than PROXIM under extreme churn. NEIGHBOR is comparable to PROXIM under light churn, but degrades under severe and extreme churn as in no replication. Next, we investigated performance sensitivity to the replication factor under severe churn (i.e. $s = 1000$). As can be seen in Figure 22(b), SELF is better than PROXIM across the replication factor. NEIGHBOR shows worse results than PROXIM under severe churn same as no replication case.

To summarize, *the proposed selection techniques consistently outperform the standard techniques in replicated environments.* The results under churn are fairly consistent with the results without replication: SELF is still better than PROXIM under severe churn, and NEIGHBOR is comparable to PROXIM under light churn.

6 Related Work

Distributed storage systems. A great deal of research on distributed storage systems has been carried out for past several years [4, 24, 18, 12]. Since they target large-scale systems, their main focus is to provide a high degree of availability and durability of data objects against independent or correlated failures [10, 28]. Despite a high probability of availability, objects can still be inaccessible to nodes in wide-area, loosely-coupled systems. Our work focuses on accessibility in order to preserve predictable data access.

Replication strategy. Several techniques have been introduced to improve data access. Some reactive or proactive replication techniques have been studied in unstructured overlays [11, 26] and structured overlays [12, 16]. These replication techniques are helpful in data access by disseminating objects in advance, but they tend to use proximity as a guiding



(a) Replication factor = 4

(b) Mean session length = 1000

Figure 22. Impact of churn under replication

principle. We have found that proximity is not sufficient to guarantee good accessibility.

Bandwidth prediction. Prediction of network bandwidth and data transfer times may also be useful to estimate data access when allocating jobs. Previous studies in [44, 13] suggested prediction methods, but their approaches were based on explicit network probing. Unlike this, we rely on local, historic observations without intrusive probing which may be expensive in loosely-coupled environments. In previous work [21], we showed how a fixed group of client nodes can select replicated data servers by a soft prediction of server capacity or load. In this paper, we consider where a client should run such that it has good availability to a data server.

Resource discovery. Resource discovery is also closely related to our work. Condor provides a matchmaking framework which provides a stateless matching service [35]. Jik-Soo et al. [22] presented a decentralized matchmaking based on aggregation of resource information and CAN (Content Addressable Network) routing [37]. The CCOF (Cluster Computing on the Fly) project [46] seeks to harvest CPU cycles by using search methods in a peer-to-peer computing environment. These resource discovery techniques focus on the specifications of individual nodes, e.g. CPU, memory, and disk space. However, they are less concerned about data access performance.

Data Grid. Finally, the Data Grid has been proposed to enable researchers to access and analyze significant volumes of data on the order of terabytes [8, 19, 36, 43, 25]. For efficient data access, the Data Grid provides integrated functionalities for data store, replication, and transfer. However all these efforts have been made under the assumption of well-organized environments where sites are managed carefully and interconnected with high bandwidth links to each other. Unlike this assumption, our intention is to accommodate such applications in loosely-coupled distributed systems where bandwidth may be less available. For this reason, we focus more on decentralization, minimal message overhead, and predictable data access.

7 Conclusion

Accessibility is a crucial concern for an increasing number of data-intensive applications in loosely-coupled distributed systems. Such applications require more sophisticated resource selection due to bandwidth and connectivity unpredictability. In this paper, we presented decentralized, scalable, and efficient resource selection techniques based on accessibility. Our techniques rely only on *local, historic* observations, so it is possible to keep network overhead tolerable. We showed our estimation techniques are sufficiently accurate to provide a meaningful rank order of nodes based on their accessibility. Our techniques outperform standard approaches and are reasonably close to the optimal selection. In particular, the self estimation-based selection approached 1.4 of optimal over time, the neighbor estimation-based selection was within 2.6 of optimal with 16 neighbors, compared to a proximity-based selection that was over 3 times the optimal. With respect to the mean elapsed time, the self and neighbor estimation-based selections were 52% and 70% more efficient respectively than proximity-based selection. We also investigated how our techniques work under node churn and showed that they work

well under churn circumstances in which nodes suffer from loss of observations. Finally, we showed that our techniques consistently outperform standard techniques in replicated environments.

References

[1] 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.

[2] D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *Proceedings of CCGRID'06*, pages 73–80, 2006.

[3] G. B. Berriman, A. C. Laity, J. C. Good, J. C. Jacob, D. S. Katz, E. Deelman, G. Singh, M.-H. Su, and T. A. Prince. Montage: The architecture and scientific applications of a national virtual observatory service for computing astronomical image mosaics. In *Proceedings of Earth Sciences Technology Conference*, 2006.

[4] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker. Total recall: system support for automated availability management. In *Proceedings of NSDI'04*, pages 25–25, 2004.

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

[6] BOINC: Berkeley open infrastructure for network computing, http://boinc.berkeley.edu/.

[7] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of SIGCOMM'03*, pages 407–418, 2003.

[8] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *J. of Network and Computer Applications*, 23(3):187–200, 2000.

[9] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *J. Parallel Distrib. Comput.*, 63(5):597–610, 2003.

[10] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *Proceedings of NSDI'06*, pages 4–4, 2006.

[11] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of SIGCOMM '02*, pages 177–190, 2002.

[12] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, pages 75–80, May 2001.

[13] M. Faerman, A. Su, R. Wolski, and F. Berman. Adaptive performance prediction for distributed data-intensive applications. Technical Report CS1999-0619, 18, 1999.

[14] 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.

[15] FreePastry, http://freepastry.org.

[16] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher. Adaptive replication in peer-to-peer systems. In *Proceedings of ICDCS'04*, pages 360–369, 2004.

[17] J. D. Guyton and M. F. Schwartz. Locating nearby copies of replicated internet servers. *SIGCOMM Comput. Commun. Rev.*, 25(4):288–298, 1995.

[18] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of NSDI'05*, May 2005.

[19] W. Hoschek, F. J. Jaén-Martínez, A. Samar, H. Stockinger, and K. Stockinger. Data management in an international data grid project. In *Proceedings of GRID'00*, pages 77–90, 2000.

[20] S. Hotz. *Routing information organization to support scalable interdomain routing with heterogeneous path requirements*. PhD thesis, 1994.

[21] J. Kim, A. Chandra, and J. B. Weissman. Exploiting heterogeneity for collective data downloading in volunteer-based networks. In *Proceedings of CCGRID'07*, pages 275–282, 2007.

[22] J.-S. Kim, B. Nam, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman. Resource discovery techniques in distributed desktop grid environments. In *Proceedings of GRID 2006*, September 2006.

[23] D. Kondo, A. A. Chien, and H. Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *Proceedings of SC'04*, page 17, 2004.

[24] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*, November 2000.

[25] Y.-F. Lin, P. Liu, and J.-J. Wu. Optimal placement of replicas in data grid environments with locality assurance. In *Proceedings of ICPADS'06*, pages 465–474, 2006.

[26] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of SIGMETRICS '02*, pages 258–259, 2002.

[27] N. Massey, T. Aina, M. Allen, C. Christensen, D. Frame, D. Goodman, J. Kettleborough, A. Martin, S. Pascoe, and D. Stainforth. Data access and analysis with distributed federated data servers in climateprediction.net. *Advances in Geosciences*, 8:49–56, June 2006.

[28] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. In *Proceedings of NSDI'06*, pages 17–17, 2006.

[29] E. Ng and H. Zhang. Predicting internet network distance with coordiantes-based approaches. In *Proceedings of IEEE INFOCOM'02*, pages 170–179, 2002.
[30] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *Proceedings of HPDC'05*, 2005.
[31] Özgür B. Akan. On the throughput analysis of rate-based and window-based congestion control schemes. *Comput. Networks*, 44(5):701–711, 2004.
[32] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose. Modeling tcp reno performance: a simple model and its empirical validation. *IEEE/ACM Trans. Netw.*, 8(2):133–145, 2000.
[33] PlanetLab, http://www.planet-lab.org.
[34] PPDG: Particle physics data grid, http://www.ppdg.net.
[35] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of HPDC'98*, page 140, 1998.
[36] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proceedings of HPDC'02*, page 352, 2002.
[37] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of SIGCOMM'01*, pages 161–172, 2001.
[38] 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.
[39] Search for extraterrestrial intelligence (SETI) project, http://setiathome.berkeley.edu.
[40] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM'01*, pages 149–160, 2001.
[41] L. Tang and M. Crovella. Virtual landmarks for the internet, 2003.
[42] Y.-M. Teo, X. Wang, and Y.-K. Ng. Glad: a system for developing and deploying large-scale bioinformatics grid. *Bioinformatics*, 21(6):794–802, 2005.
[43] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling e-science applications on global data grids: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(6):685–699, 2006.
[44] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.
[45] 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.
[46] D. Zhou and V. Lo. Cluster computing on the fly: resource discovery in a cycle sharing peer-to-peer system. In *Proceedings of CCGRID'04*, pages 66–73, 2004.