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
{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. We introduce the notion of accessibility *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 show 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. The suggested techniques are also shown to be stable even under churn despite the loss of prior observations.*

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 [1], distributed storage systems [8], and Grids [11]. However, a major challenge in such systems is the network unpredictability and limited bandwidth available for data dissemination. For instance, the BOINC project 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 [1]. 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 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. For example, in high energy physics applications such as the Large Hadron Collider (LHC), thousands of physicists worldwide require access to shared, immutable data on the scale of petabytes [9]. Similarly, in the area of bioinformatics, a set of gene sequences could be transferred from a remote database to enable comparison with input sequences [2]. In these examples, performance depends critically on efficient data delivery to the computational nodes. The efficiency of data delivery for such applications would critically depend on the location of data as well as 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 distributed systems. 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 does not capture 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 smaller download time compared to the other. Therefore, in the context of data-intensive appli-

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

cations, it is important to consider *data accessibility*: how efficiently a node can access a given data object in the system.

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 rules 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 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 poor 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. First, 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 a 0.5 relative error in live experimentation on PlanetLab. Second, we present accessibility-aware resource selection techniques based on our estimation functions and compare to the optimal and standard techniques such as latency-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.

2. 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 [3, 13] and structured [21, 19, 20], or any other techniques. However, we assume that the 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 [14, 11, 23] to determine the set of candidate nodes, though it may not consider data locality in its choice. 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 $R_i = \{r_i^1, r_i^2, ...\}$ 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.

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 order 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 the candidate cannot provide any meaningful estimate of its accessibility to the required data object. In this case, the candidate simply returns a value of ∞, indicating the lack of any information. The initiator would filter out such a candidate. If all candidates return ∞, one of the candidates is *randomly* selected. The selection heuristic is a mapping function of $H_s : C \rightarrow c_m$ such that $accessibility_{c_m}(J_i) = \min_{n=1,..,|C|}(accessibility_{c_n}(J_i))$.

Having described the accessibility-based resource selection process, the question is how the candidate nodes can estimate their accessibility using local information (e.g., their own observations to the object if known or their neighbors), and what factors they can use for this estimation. We explore this question in the next section.

3. Accessibility Estimation

To answer the above question, we performed a rich set of exploration to see what parameters would impact accessibility in terms of data download time. We report important correlations we found in the experiments with 133 nodes on PlanetLab [17]. Our intuition was that 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 capabilities*, such as its connectivity and bandwidth. As expected, we find a negative correlation ($r = -0.56$) between RTT and download time, indicating that latency can be a useful factor when estimating accessibility between node pairs. In addition, we discovered a correlation ($r = 0.6$) between the download speed of a node and the past average download speed of the node, suggesting that the past download behavior of a node can be a useful component for accessibility estimation.

Based on the statistical correlations we discovered, we next present estimation techniques to predict data access capabilities of a node for a data object by utilizing local information.

3.1. Self-Estimation

As described above, latency to the server[1] 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 the following information: object name, object size, download elapsed time, server, distance to server, and timestamp.

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 [16] and rate-based [15] 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 thus far is then calculated by $\overline{Distance_h} = \frac{1}{|\mathcal{H}_h|} \cdot \sum_{i=1}^{|\mathcal{H}_h|} \mathcal{H}_h^i.distance$.

[1]For ease of exposition here, we assume each data object is located on a single server. However, we relax this assumption and consider data replication in our experiments in Section 4.7.

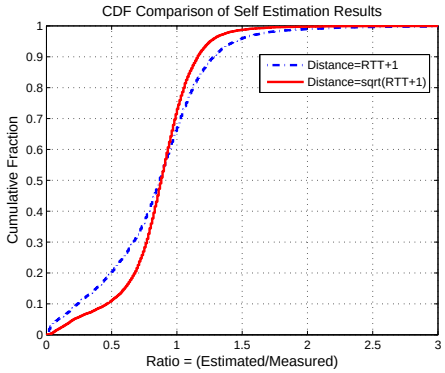

Figure 1. Self estimation result

We then characterize the network characteristics of the node by estimating its mean download speed based on prior observations. The mean download speed of node h is defined as $\overline{DownSpeed_h} = \frac{1}{|\mathcal{H}_h|} \cdot \sum_{i=1}^{|\mathcal{H}_h|} \frac{\mathcal{H}_h^i.size}{\mathcal{H}_h^i.elapse}$.

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

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

where $\delta = \frac{distance_h(server(o))}{\overline{Distance_h}}$, $size(o)$ 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 is the ratio of the distance to the server for object o to the mean distance to all servers it has observed thus 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 1 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 fall within a ratio of 0.5 – 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]. We will see in Section 4 that this level of

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

accuracy is sufficient for use as a ranking function to order different candidate nodes for resource selection.

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 [3, 13], or by routing messages in structured overlays [21, 19, 20], which may introduce extra traffic. Distance determination would require probing which adds additional overhead.

3.2. Neighbor Estimation

While self-estimation uses a node's prior observations (to all objects) 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 get a more accurate estimation. These will 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 both search and probing, thus avoiding their costs. 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 host h as follows:

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

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. This is because if two clients show the same download time for the same object, the more distant one might be considered to have better downloading capability for more distant servers, as the closer client's download speed could be attributed to its locality. Hence, access over greater distance is given greater weight in this metric.

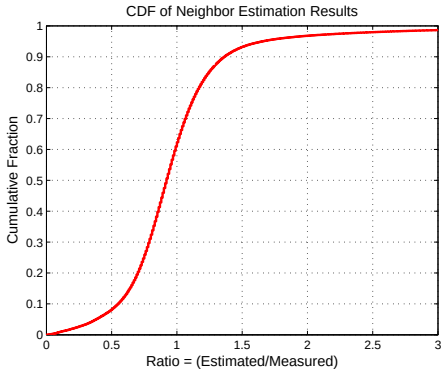

Figure 2. Neighbor estimation result

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

$$NeighborEstim_h(n, o) = \alpha \cdot \beta \cdot elapse_n(o) \quad (3)$$

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[3].

Intuitively, to estimate the download time for object o based on the information from neighbor n, this function 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.

Figure 2 illustrates the cumulative distribution of neighbor estimation results. The x-axis is the ratio of the estimation result to the real measured value, while the y-axis is the cumulative 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 of 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 neighbors. Thus it is possible that multiple neighbors provide different information for the object,

[3] 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.

Table 1. Download Traces

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

thus yielding different estimation results. In this case, we simply take the median of the estimations based on our experiments (see [10]). In addition, we can avoid the need for active probing to measure the latency to the server by exploiting the server latency estimates obtained from the neighbors themselves. In [10], we extend latency inference heuristics to more accurately work with a *limited* number of neighbors.

4. Evaluation

4.1. 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 [20, 6] 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. 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.

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[4]. After constructing the network, the simulator runs each resource selection algorithm. At first, 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. As the default configuration, we set both the candidate size and the neighbor size to 8, unless otherwise mentioned.

[4]To minimize error due to the construction, we repeated simulations 50 times and report the results with 95% confidence intervals as needed.

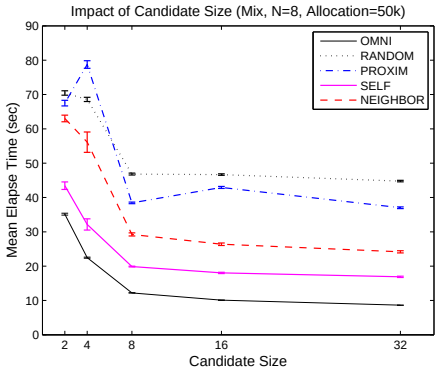

Figure 3. Impact of candidate size

For our evaluation, we compare the following resource selection techniques:

Technique	Selection
OMNI	omniscient selection (optimal)
RANDOM	random selection
PROXIM	latency-based selection
SELF	self-estimation-based selection
NEIGHBOR	neighbor-estimation-based selection

4.2. Impact of Candidate Size

In our system model, a set of candidate nodes are evaluated for their accessibility before allocating a job. We investigate the impact of candidate size ($|C|$). Figure 3 demonstrates the mean downloading time with respect to candidate size. 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. Interestingly, PROXIM shows unstable results with greater fluctuation than RANDOM over the candidate sizes. 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*.

4.3. Impact of Neighbor Size

We next investigate the impact of neighbor size on NEIGHBOR (the other heuristics are not affected by this parameter). Figure 4 shows how the selection techniques respond across the number of neighbors ($|N|$). As can be seen, 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 $|N| = 2$. This is because it has more chances to obtain relevant observations with more many neighbors, thus decreasing the possibility

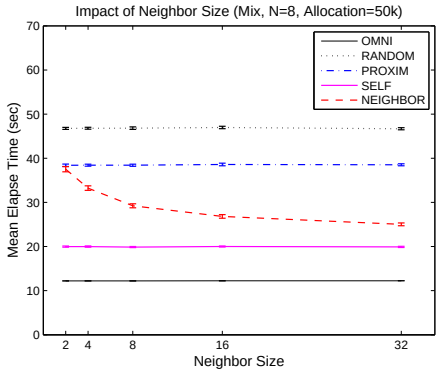

Figure 4. Impact of neighbor size

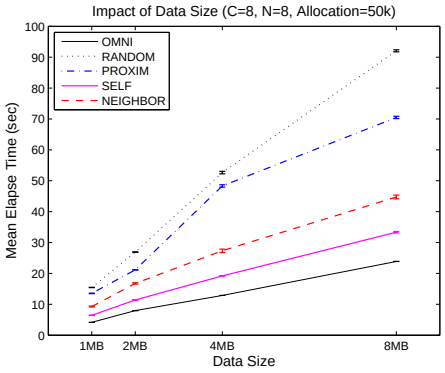

Figure 5. Impact of data size

of random selection. This result suggests that *NEIGHBOR will work better in environments where the node has connectivity with a greater number of neighbors*.

4.4. 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, 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 5 shows linear relationship between data size and mean download time. However, we can see that each technique shows a different degree of slope: SELF and NEIGHBOR increase relatively gently compared to 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. This result implies that *the proposed techniques not only work consistently across different data sizes, but they are also more useful for data-intensive applications*.

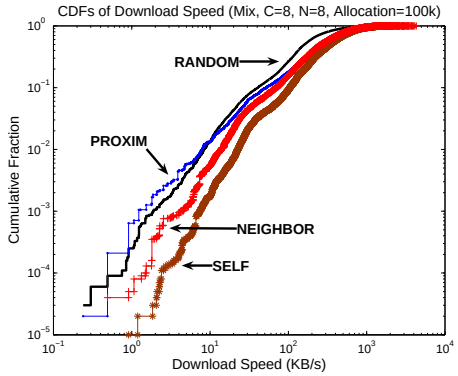

Figure 6. Timeliness

4.5. 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 a good node every time because there are many contributing factors.

We observed how many times the techniques choose slow connections. Figure 6 shows cumulative distributions of the speed of connections with log-log scales. In the figure, we can see that the proposed techniques more often avoid slow connections. SELF most successfully excludes low speed connections, and NEIGHBOR also performs well compared to 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 greater 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 latency information alone greatly increases the chance of very poor connections, thus leading to unpredictable response time*. Compared to this, our proposed techniques successfully reduce chances to choose low speed connections by taking accessibility into account.

4.6. 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

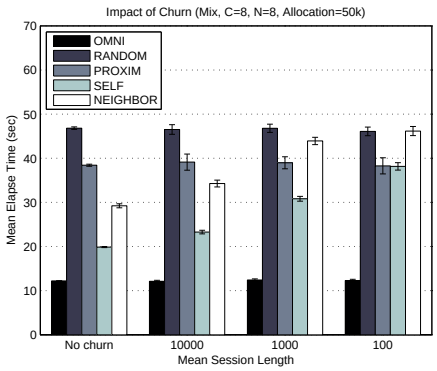

Figure 7. Performance under churn

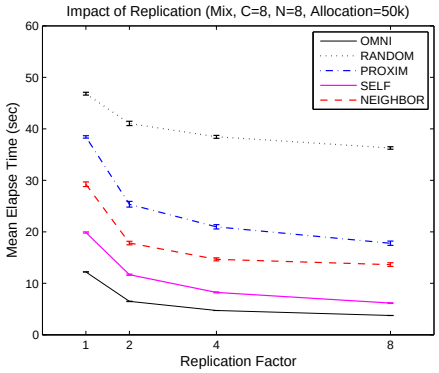

Figure 8. Performance under replicated environments

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.

Figure 7 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. As can be seen in the figure, SELF outperforms or is comparable to PROXIM. 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. Nonetheless, NEIGHBOR still works better than PROXIM in light churn ($s = 10000$) with lower overhead. To summarize, *the proposed techniques are fairly stable under churn in which nodes suffer from loss of observations*. The results show that SELF is at least comparable to PROXIM, while NEIGHBOR is comparable to PROXIM when churn is light.

4.7. 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 work consistently in replicated environments. For lack of space, we briefly report the experimental results, and the details can be found in a technical report [10].

Figure 8 shows performance changes under replication. 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 the figure. PROXIM has significantly diminished mean download time (nearly half) with replication. However, it is still behind the proposed techniques. SELF and NEIGHBOR outperform the standard techniques over all the replication factors. In particular, SELF continuously narrow the gap to optimal favorably exploiting the replication. This result suggests that *the proposed selection techniques consistently outperform the standard techniques in replicated environments.*

5. Related Work

Several techniques have been introduced to improve data access. Some reactive or proactive replication techniques have been studied in unstructured overlays [3, 13] and structured overlays [4, 7]. These replication techniques are helpful in data access by disseminating objects in advance, but they tend to use proximity as a guiding principle. We have found that proximity is not sufficient to guarantee good accessibility.

Prediction of network bandwidth and data transfer times may also be useful to estimate data access when allocating jobs. Previous studies in [22, 5] 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.

Resource discovery is also closely related to our work. Condor provides a matchmaking framework which provides a stateless matching service [18]. Jik-Soo et al. [11] presented a decentralized matchmaking based on aggregation of resource information and CAN (Content Addressable Network) routing [19]. The CCOF (Cluster Computing on the Fly) project [23] 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.

Finally, the Data Grid has been proposed to enable researchers to access and analyze significant volumes of data on the order of terabytes [9, 12]. 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.

6. 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 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 and neighbor estimation-based selections were 52% and 70% more efficient respectively than latency-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 and G. Fedak. The computational and storage potential of volunteer computing. In *Proceedings of CCGRID'06*, pages 73–80, 2006.

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

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

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

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

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

[7] 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.

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

[9] 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.

[10] J. Kim, A. Chandra, and J. B. Weissman. Accessibility in loosely-coupled distributed systems. Technical Report 07-028, University of Minnesota, November 2007.

[11] 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.

[12] 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.

[13] 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.

[14] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *Proceedings of HPDC'05*, 2005.

[15] Özgür B. Akan. On the throughput analysis of rate-based and window-based congestion control schemes. *Comput. Networks*, 44(5):701–711, 2004.

[16] 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.

[17] PlanetLab, http://www.planet-lab.org.

[18] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of HPDC'98*, page 140, 1998.

[19] 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.

[20] 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.

[21] 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.

[22] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[23] 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.