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 08-041

OPEN: Passive Network Performance Estimation for Data-intensive Applications

Jinoh Kim, Abhishek Chandra, and Jon Weissman

November 24, 2008

OPEN: Passive Network Performance Estimation for Data-intensive Applications

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

Distributed computing applications are increasingly utilizing distributed data sources. However, the unpredictable cost of data access in large-scale computing infrastructures can lead to severe performance bottlenecks. Providing predictability in data access is thus essential to accommodate the large set of newly emerging large-scale, data-intensive computing applications. In this regard, accurate estimation of network performance is crucial to meeting the performance goals of such applications. Passive estimation based on past measurements is attractive for its relatively small overhead compared to relying on explicit probing. In this paper, we take a *passive approach* for network performance estimation. Our approach is different from existing passive techniques that rely either on past direct measurements of pairs of nodes or on topological similarities. Instead, we exploit secondhand measurements collected by other nodes without any topological restrictions. OPEN (Overlay Passive Estimation of Network performance) is a scalable framework providing end-to-end network performance estimation based on secondhand measurements. In this paper, we present extensive experimental results, including simulation with our PlanetLab trace data sets and S^3 bandwidth measurement data sets and a live experiment with Montage, a toolkit for astronomical research. The experimental results show that OPEN consistently outperforms selection techniques based on statistical pairwise estimations as well as random and latency-based selections in diverse experimental settings.

1 Introduction

In the distributed computing domain, demands on data have significantly increased over the past few years, and importantly, such applications are increasingly utilizing *distributed* data sources. For example, projects such as LHC [28] in high energy physics and SkyServer [41] in astronomy produce petabytes of new data every year, and researchers over the world access the data, often distributed, for their own experiments. An example of such applications is data-intensive scientific workflows [5, 10]. For such data-intensive tasks, data access cost is a significant factor in their execution performance in addition to the computation cost. Hence, it is essential to consider data access cost in launching data-intensive computing applications.

Large-scale computing infrastructures such as grids [13, 44] and desktop grids [2, 25, 29] are attractive due to their scalability and cost-effectiveness. However, the loosely-coupled nature of many of these platforms often makes them unpredictable in their resource availability and performance, particularly in terms of data access. Despite their rich set of computational resources, the unpredictable nature of large-scale computing platforms makes it hard to deploy such data-intensive applications or limits the size of data access making them inefficient to deploy. Thus, providing predictability in data access is a vital requirement for enabling such data-intensive tasks on large-scale systems. The goal of this work is to successfully execute data-intensive computing applications on unpredictable but appealing large-scale systems.

A key requirement for achieving data access predictability is the ability to estimate network performance for data transfer, so that computation tasks can take advantage of the estimation in their deployment or data

source selection. In other words, network performance estimation can provide a helpful guide to run data-intensive tasks in such unpredictable infrastructures having a high degree of variability in terms of data access.

Active probing can be an option for estimation, but is unscalable and expensive in using back-to-back measurement packets. Passive estimation is attractive for its relatively small overhead, and thus could be desirable for many networked applications that do not require an extremely high degree of accuracy such as that needed by network-level applications like network planning. For example, a substantial number of networked applications, such as Web server selection and peer selection for file sharing, rely on *ranking*. According to a peer-to-peer measurement study in [33], the second placed peer performance is only 73% of the best peer performance. This significant gap implies that some degree of estimation inaccuracy would be tolerable for such ranking-based applications. A potential problem of passive estimation is that it can suffer from estimation failure due to the unavailability of past measurements. This problem can be mitigated by sharing measurements among nodes; thus, a node can estimate performance even against a server it has never contacted. In previous work [3, 39], however, the sharing was restricted to specific underlying topologies such as a local network, limiting scalability. In this work, we present a novel approach enabling nodes to utilize past measurement information with no reliance on topological similarities, so as to minimize blind spots in the system and to reduce uncertainty in data access.

To realize this goal, there are two important challenges. The first challenge is the *characterization* of a node in terms of its data access capability to enable it to utilize others' measurements for its own estimation. This characterization is key for topology-independent utilization of secondhand measurements. The other important challenge is how to facilitate local measurements to be globally available to other nodes in the system for system-wide sharing. Any server-based techniques for storing global information are limited by well-known problems of scalability and fault tolerance. At the other end of the spectrum is flooding-based dissemination, which while fully distributed, has high network overhead. In this paper, we present *OPEN (Overlay Passive Estimation of Network performance)*, a scalable framework for end-to-end network performance estimation. OPEN provides a correlation-based secondhand estimation with empirical node characterization (proposed in our previous work [24]) and *proactive dissemination* of measurements with limited overhead by diverse optimizations.

Our key contributions can be summarized as follows:

- We present the OPEN framework that performs passive network performance estimation based on secondhand downloading information. OPEN is highly scalable, fully distributed, and topology-neutral.
- To enable cost-effective information sharing, we introduce two optimization techniques in addition to probabilistic approach, both of which separately disseminate measurement information based on its "criticality", i.e., how important it is to share in the system. We show these optimization techniques dramatically reduce dissemination overhead without significant performance loss.
- We evaluate OPEN with two selection problems common in distributed computing, resource selection and replica selection. To emulate a large-scale system, we collected actual download traces for 10 months in PlanetLab [36]. Using data sizes contained in GridFTP workloads [26], the results show that OPEN consistently outperforms selection techniques based on statistical pairwise estimations as well as random and latency-based selections in diverse experimental settings.
- We deploy our OPEN framework in PlanetLab and evaluate with Montage [4], a toolkit for astronomical research. We run a Montage application on the OPEN framework and present performance results.

2 Distributed Computing Model

We consider a large-scale infrastructure for distributed computing. The system consists 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 that both compute nodes and data nodes are connected in an overlay structure without any assumption of centralized entities for scalability. We do not assume any specific type of organization for the overlay, but assume that the overlay provides basic data access functionalities including *search*, *store*, and *retrieve*. In addition, each node in the system can be a compute node, data node, or both.

Figure 1 illustrates the distributed computing model we consider. In the worker pool, computational resources are provided to run applications, while data

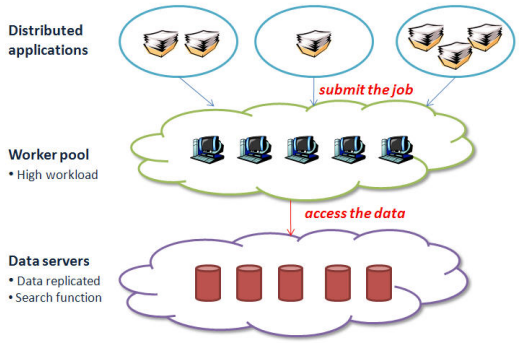


Figure 1. Distributed computing model

nodes serve data objects accessed by the compute nodes. Distributed applications share the computational resources by submitting their jobs. In this model, we focus on two selection problems common in the distributed computing domain.

- *Replica selection*: The data object is replicated in multiple data nodes geographically dispersed, and the compute node needs to select a replica to download. The goal of this selection is to identify a replica server having minimal data access cost from the compute node.

- *Resource selection*: For job allocation, one or more compute nodes should be chosen from a list of computational resources. In this context, the job requires accessing data for task completion. The goal of this selection is to identify a compute node which can access the data server with minimal data access cost.

Formally, the worker pool W consists of compute nodes, $W = \{w_1, w_2, ..\}$, while the data server pool consists of data nodes, $S = \{s_1, s_2, ..\}$. Data objects can be replicated in a set of data nodes, $R_i = \{r_1, r_2, ..\}$, where $R \subseteq S$. Replica selection picks the replica nodes with minimal estimated data access cost to the object. We define $cost(a, b)$ as the data access cost between client a and server b. Hence, replica selection is a function (H_1) to choose the minimal cost: $H_1(R) \in R$ s.t. $cost(c, H_1(R)) \leq cost(c, r)$, for all $r \in R$.

For resource selection, we are given job J, which needs to access a data object replicated to a set of data nodes R, and a set of candidate nodes to assign the job, $C = \{c_1, c_2, ..\}$, where $C \subseteq W$. This candidate set can be determined by a centralized scheduler [8, 37], a resource discovery algorithm [7, 22, 34, 50], or any other directory services. Here, the resource selection problem is to select the candidate node with the minimal estimated data access cost to the required object. Similar to the replica selection function (H_1), resource selection is a function (H_2) to choose the minimal cost compute node: $H_2(C) \in C$ s.t. $\min_{r \in R}(cost(H_2(C), r)) \leq \min_{r \in R}(cost(c, r))$, for all $c \in C$.

Hence, the *cost of data access* is a vital factor for both selection problems. While the computation cost is also important for overall task performance, our focus in this paper is on the communication cost. The question is *how to estimate the data access cost accurately and cheaply for the selection problems described above.* In next section, we discuss estimation techniques based on past firsthand or secondhand measurement information and the benefits of secondhand measurement-based estimation.

3 Secondhand Estimation

We classify estimation techniques into three categories, based on the *degree of measurement sharing* for their estimation: *pair-level*, *domain-level*, and *system-wide*, as summarized in Table 1. Pair-level sharing only utilizes the direct (*firsthand*) measurements made by a *specific pair* of nodes for their network path estimation. Many statistical or time-series forecasting techniques, such as exponential moving average, belong to this class. Previous studies, such as [46], showed the high accuracy of these techniques, but this class requires $O(n^2)$ measurements for estimation between all pairs.

In contrast, some estimation techniques enables nodes to utilize indirect (*secondhand*) measurements provided by other nodes for their own estimation. In domain-level sharing, past measurements in a domain (e.g., a single network or logical group of nodes) are shared between nodes belonging to the same domain. In SPAND [39], nodes in a single network share past measurements for Web server selection. Webmapper [3] shares passive measurements to select a Web server based on a logical group clustered by IP prefixes. By sharing the measurements in a domain, it is possible to estimate performance if any node in the domain has communicated with the server. Again, however, the sharing is restricted to the domain. In addition, the underlying assumption of existing techniques belonging to this class is that the nodes in a domain have closely similar characteristics in network access. If this is not the case, sharing measurements without considering node characteristics may cause inaccuracy in estimation.

Unlike the above two classes of sharing, system-wide sharing, we propose in this work, has no constraints

Table 1. Degree of measurement sharing

Degree	Pair-level	Domain-level	System-wide
Approach	Statistical estimation Time-series forecast	Sharing in a LAN Sharing in a domain	Sharing in a system
System/ Technique	NWS [46] HB prediction [17]	SPAND [39] Webmapper [3]	OPEN

on sharing measurements across the system. In other words, if any measurement against a server is available in the system, any other node can utilize that information for its own estimation to that server. Thus it is possible to perform any-pair estimation with $O(n)$ measurements. Since it does not rely on topological similarities, node characterization is essential to utilize others' experiences. In addition, efficient sharing is also a key for this approach. Before discussing how OPEN realizes those key functions, we briefly describe the rationale for secondhand estimation in large-scale infrastructures.

3.1 Why Secondhand Estimation?

While firsthand-based estimation is likely to be more accurate than secondhand-based estimation, it is unlikely that all nodes will have firsthand observations to all servers (a worst-case of $O(n^2)$ total measurements in the system if all workers are also data servers). Thus, there would be no estimates available for node pairs that lack direct measurements.

Figure 2 compares the potential estimation failure rates of a pairwise firsthand estimation technique to that of a system-wide secondhand estimation approach (OPEN)[1], caused by a shortage of existing relevant measurements. This result is obtained through a trace-driven simulation[2], where we tested 100,000 estimations in two systems with size $n = 100$ and $n = 1000$. We assume there are no measurements at all in the beginning, and one random pairwise measurement is recorded at each time instant. As can be seen from the figure, the failure rates decrease as more measurements are added over time. In particular, we observe that OPEN dramatically diminishes the failure rates over time by using secondhand measurements for estimation. In contrast, the pairwise firsthand technique suffers from significant failure rates; the system with $n = 1000$ has over 90% failure, even at the end of the simulation. This is because the probability that a node has any measurements to a server

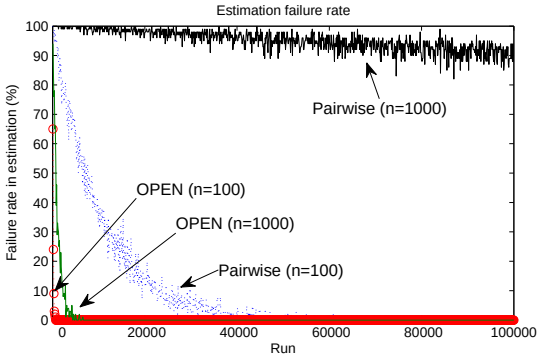


Figure 2. Hit rate of relevant measurements

goes down as the system size grows. Given that a large-scale system can consist of tens of thousands nodes, the pairwise approach must ensure, in the worst case, that $O(n^2)$ measurements exist, which could require active probes to fill in the gaps due to insufficient firsthand observations; or it may suffer from high failure rates due to a lack of sufficient measurements. Therefore, the secondhand approach should be beneficial in terms of both scalability and overhead.

Again, domain-level sharing also performs secondhand estimation, but relies on topological similarity. Our intention is to design a framework to enable secondhand estimation without any topological constraints, as described in the next section.

4 The OPEN Framework

In our previous work [24], we proposed an estimation technique utilizing measurements observed in the neighbor nodes in an overlay network. While working well in a small setting, the estimation technique can suffer from a shortage of measurements in a large-scale environment, significantly degrading performance. In this work, we relax the constraint on measurement sharing. The OPEN framework we present in this paper enables system-wide sharing of measurements with-

[1] OPEN uses dissemination of secondhand measurements, as will be discussed in more detail in Section 4.2.

[2] We will present details of the trace and our methodology in Section 5.1.

Table 2. Attributes of measurement

Attribute	Description	Which record
id	Unique ID	Both ($\mathcal{L}, \mathcal{I}$)
client	Measurement node	Imported ($\mathcal{I}$)
server	Data server	Both ($\mathcal{L}, \mathcal{I}$)
distance	Distance to server	Both ($\mathcal{L}, \mathcal{I}$)
throughput	Measurement throughput	Both ($\mathcal{L}, \mathcal{I}$)
DP	Download power	Imported ($\mathcal{I}$)
timestamp	Time stamp	Both ($\mathcal{L}, \mathcal{I}$)

out both underlying and overlay topological restrictions. OPEN provides end-to-end network performance based on shared secondhand measurements, unlike the previous work providing the expected downloading time for a specific data object. In this section, we present the OPEN framework with respect to two core mechanisms: *secondhand estimation* of end-to-end network performance and *proactive dissemination* of observed measurements. We now discuss how OPEN implements its functionality.

4.1 Passive estimation

OPEN utilizes two types of measurements for estimation for a node: *local measurements* ($\mathcal{L}$) measured directly by the node, and *imported measurements* ($\mathcal{I}$) obtained from other nodes. OPEN makes an estimation by comparing and combining the node capability in data access from its local measurements to the imported measurements of other nodes as we will show. Table 2 summarizes the attributes defined in the local and imported measurement records. We use *dot*(.) notation to refer to each attribute in the record, and the cardinality (e.g., $|\mathcal{L}|$ and $|\mathcal{I}|$) represents the number of records stored in the node.

We first characterize a node based on the local measurements at a node. As introduced in our previous work [24], *download power (*DP*)* is a quantitative metric empirically formulated to characterize a node (h) based on prior local data access behaviors:

$$DP_h = \frac{1}{|\mathcal{L}_h|} \sum_{i=1}^{|\mathcal{L}_h|} (\mathcal{L}_h^i.throughput \times \mathcal{L}_h^i.distance) \quad (1)$$

The above equation implies that a greater throughput and distance produce a greater characterized metric. Thus the node with a greater DP is considered better in data access capability. Intuitively, throughput has a degree of correlation with distance; for example, we could expect a higher throughput for a closer server if all other conditions are the same. By combining those two factors in Equation 1, the download power provides a node capability in terms of data access independent from the distance factor; thus we can compare this metric without considering the distance to server. As seen from Equation 1, $DP \propto$ *throughput* 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.

With the characterized metric DP, we compute similarity between two nodes i and j by the following equation:

$$\mathcal{S}(i, j, s) = \frac{DP_i}{DP_j} \cdot \frac{distance(j, s)}{distance(i, s)} \quad (2)$$

The scaling factor $\mathcal{S}$ is used to compare the download characteristics of any two unrelated nodes in the system to enable the appropriate scaling of imported measurements for estimation. $\mathcal{S}(i, j, s) = 1$ means that two nodes i and j are exactly the same with respect to data retrieval from server s. If the scale value = 2, it means that the node has a factor of two capability in accessing given server. Hence, $\mathcal{S}(i, j, s) < 1$ indicates that node i is inferior to node j in accessing server s, and vice-versa. In the equation, $distance(a, b)$ refers to the distance metric from a to b.

Based on the scaling factor, OPEN produces the estimate by utilizing the imported measurements from nodes to the same server. The following equation is used to estimate the expected throughput (T) between a node (h) and a server (s) with an imported measurement m:

$$T_{h,s} = \mathcal{S}(h, m.client, s) \times m.throughput \quad (3)$$

It is possible that there exist multiple imported measurements to the same server. To combine multiple estimates in such cases, we investigated a rich set of statistical techniques and observed that the median performs best.

To evaluate accuracy of the OPEN estimation, we performed simulation with actual downloading traces (summarized in Table 3). In the simulation, 10,000 estimations were made by Equation 3, and we computed *relative error*, which is widely used to evaluate accuracy of estimation [9, 32, 43, 49]. Relative error (RE) is computed by:

$$RE = \frac{|\textit{estimated value} - \textit{measured value}|}{\min(\textit{estimated value}, \textit{measured value})}$$

Thus, it implies perfect estimation when the relative error is zero, while the relative error 1 means that the

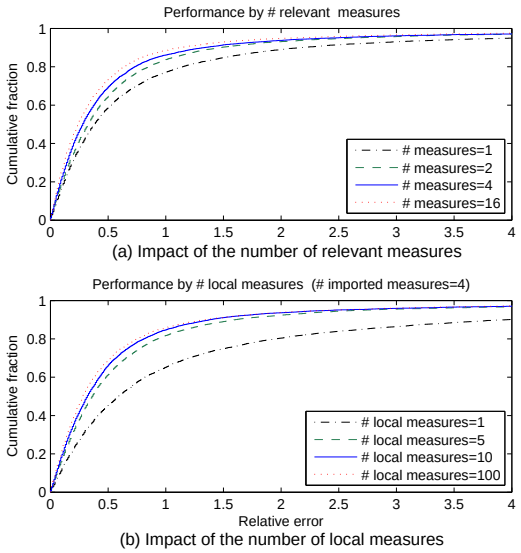


Figure 3. Relative error of OPEN estimates

estimated value is a factor of two, either smaller (underestimation) or greater (overestimation), of the measurement value.

Figure 3 illustrates the cumulative distribution of the OPEN estimation results. The x-axis is relative error of estimates. Figure 3(a) shows the impact of the number of secondhand measurements, while Figure 3(b) shows the impact of the number of local measurements when the number of secondhand measurements is 4. As can be seen in Figure 3(a), the estimation with 4 secondhand measurements approximates to the estimation results with the greater number of measurements, yielding roughly 90% of estimations located within a factor of two. The accuracy drops quickly with a single secondhand measurement. In Figure 3(b), we can see that the estimation with only one local measurement poorly performs. However, it performs quite well with 5 local measurements. It continuously improves with 10 local measurements, but does not further improve with more than 10. These results indicate that *the OPEN framework enables nodes to participate in estimation without a costly learning phase.* In addition, it implies that storage requirements can be small.

We add comments on estimation accuracy. Of past estimation work, Spruce [43] is a pairwise bandwidth estimation tool based on packet pairs. In their Internet experiments, 70% of the estimations are located within relative error 0.3, and roughly 80% and 90% are located within relative error 0.5 and 1 (i.e., a factor of two), respectively. In GNP [32], a network coordinate system, the best result in estimating latency is that approximately 90% and 95% are located within relative error 0.5 and 1, respectively. Our results show that 60% of the estimates lie within relative error 0.3, and 77% and 89% are in relative error 0.5 and 1, respectively. The OPEN estimation accuracy is slightly below Spruce's, however this is expected as OPEN is not a pairwise estimation technique utilizing firsthand measurements (as is Spruce). Nonetheless, we see that the number of estimates less than RE 0.5 in OPEN is comparable to Spruce.

In Equation 3, all the terms except $distance(h, s)$ can be obtained from past measurements. Since we compute distance from end-to-end latency, it is possible to employ a lightweight latency prediction technique, such as network coordinate systems [9, 32]. For example, Vivaldi, which is also used in the SWORD resource discovery tool [34], predicts end-to-end latency based on piggybacking, thus minimizing explicit probing.

Since there is no topological dependence in this estimation process, any node can utilize secondhand measurements any other nodes experienced. Thus the next question is *how we can efficiently share measurements across the system.*

4.2 Proactive dissemination

In this section, we discuss how OPEN can achieve proactive dissemination with limited overhead, but without significant performance loss.

4.2.1 Probabilistic dissemination

The simplest form of dissemination is to immediately forward new information to the neighbors at every node. This would be helpful for quick propagation of the information, but such a flooding can critically disrupt user traffic, degrading overall system performance.

The probabilistic approach can reduce such dissemination overhead by forwarding the information to a partial set of neighbors (instead of all the neighbors). In this technique, *dissemination probability* (p) defines the probability that a node forwards disseminated information to neighbor nodes. Thus $p = 1$ is equivalent to flooding, while $p = 0$ means no dissemination at all. Once a node generates (or receives) new information, it forwards the information based on p, thus the average fanout is approximately $\lceil p \times |N| \rceil$, where $|N|$ is the number of neighbors of the node. While perfect in dissemination, flooding makes a huge number of duplicate dissemination messages for the same information. By

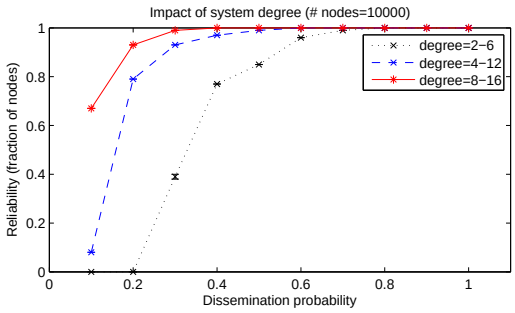


Figure 4. Impact of system degree on dissemination reliability

assigning $p < 1$, thus, it is possible to reduce such duplications.

As one can imagine that higher dissemination probability increases the number of dissemination messages, we observed a linear relationship between the probability and the number of messages. One interesting result is that system degree (i.e. average neighbor size in the system) has considerable impact on dissemination reliability, a metric to represent what fraction of nodes successfully received disseminated information. Figure 4 presents dissemination reliability with respect to dissemination probability in 3 systems with different system degrees. The probabilistic dissemination opens up a rich space of optimizations in the OPEN framework. It is relatively straightforward to determine the dissemination probability for systems having deterministic degree; for example, many structured overlays [38, 42] make use of predetermined neighbor size. Other systems with non-deterministic degrees may require a discovery mechanism to identify an adequate dissemination probability for the system. This should take into account that the topology and node degree could change dynamically. Discovering adequate dissemination probability for such non-deterministic systems is one of our future research areas.

One more optimization to reduce dissemination overhead would be *periodic* dissemination. Unlike immediate forwarding, periodic dissemination holds new information for a certain time period, thus enabling redundant information to be compressed. Then accumulated information is delivered at each time interval. Since many overlay networks employ periodic heartbeats for health check between neighbors, periodic dissemination could be realized via piggybacking.

Probabilistic dissemination opens up a rich space of optimizations to the OPEN framework. In addition to such probabilistic optimizations, OPEN provides further optimizations based on "criticality" of the measurement, i.e. whether it is highly important (or *hot*) to the system or relatively less important (or *cold*). We next introduce two optimization techniques called *selective eager dissemination*, which disseminates hot information eagerly but cold information periodically, and *selective deferral and release*, which defers distribution unless the measurement is determined to be hot within a time-bound.

Algorithm 1 Selective eager dissemination

```
1:  initiate(message m):
2:  if is_eager(m) == true then
3:      forward(m);
4:  else
5:      forwardList.append(m);
6:  end if

7:  receive(message m):
8:  if message ∉ historyList then
9:      historyList.append(m);
10:     if is_eager(m) == true then
11:         forward(m);
12:     else
13:         forwardList.append(m);
14:     end if
15: end if

16: timeout():
17: forward(forwardList);
18: forwardList ← ∅

19: forward(message_array m[]):
20: N ← neighbor nodes;
21: for all n ∈ N do
22:     if random() ≤ p then
23:         send m[] to n;
24:     end if
25: end for
```

4.2.2 Selective eager dissemination

Although periodic dissemination can greatly diminish the number of dissemination messages, one limitation with this technique is the propagation delay due to its periodicity. Some applications need to spread critical information more quickly. For example, we may want to disseminate the secondhand measurement if we have no information about *that* server in the measurement yet, in order to reduce the potential miss rates in estimation. To handle this, we consider *selective eager dissemination*, which disseminates hot information quickly without delay, while cold information is delivered periodically. In other words, only critical information is *eagerly* propagated to the system in this technique.

Algorithm 1 illustrates the procedure of selective eager dissemination. Function **initiate** is performed by

a source node when a new measurement is obtained by actual downloading, and the source node determines if the new measurement is worth being distributed *eagerly*. Based on the decision, the measurement is either forwarded to neighbors at once (if *is_eager(m) == true*) or stored in the list for periodic dissemination (if *is_eager(m) == false*). A receiving node performs a similar function: it forwards the information immediately if it is hot; otherwise, it is moved to the periodic forwarding list, as seen in the **receive** function. Each node performs periodic dissemination when the periodic timer expires by the **timeout** function. The internal functions can be defined on the local state, as perceived by the initiating or receiving node.

Hot information can be determined in several ways, such as by using repetitive counters, timestamps, statistical deviations, or any combination of these techniques. However, determining hot information is application specific. In this work, we use a *threshold*, such that if the number of measurements for a server is below this threshold, then the server-specific measurement is more eagerly distributed. For example, if a measurement is "below-threshold," then a node would forward it without any delay; otherwise, the measurement is regarded as cold. Thus, it is lazily forwarded after the given periodic interval expires.

4.2.3 Selective deferral and release

Another optimization technique we introduce is selective dissemination based on *deferral and release conditions*, which define whether new information can be deferred (for its dissemination) or released (to the system). If a "deferral" decision is made for some new information, the source node does not emit it into the system until the corresponding "release" condition is met. Thus, the deferral condition tests if new information is critical, while the release condition retests if deferred information is critical based on the passage of time. In this technique, any deferred information will either be disseminated if it becomes important later or discarded when it becomes stale. In contrast, selective eager dissemination ultimately forwards all information.

The basic idea of this technique is to distribute a newly collected measurement only if it offers unique information different from past measurements. For example, suppose node A makes an estimation of 100KB/s for end-to-end throughput to node B based on past shared measurements. Now assume node A just downloaded a data object from B with 100KB/s throughput. Then node A may not want to disseminate such redundant information to others (*deferral*). However, this cold information

Algorithm 2 Selective deferral and release

```
 1: initiate(message m):
 2: if deferral_cond(m) == true then
 3:     deferredList.append(m);
 4: else
 5:     forward(m);
 6:     release_test(m);
 7: end if

 8: receive(message m):
 9: if message ∉ historyList then
10:     historyList.append(m);
11:     forward(m);
12:     release_test(m);
13: end if

14: release_test(message m):
15: D ← deferred messages to the same server as m from deferredList;
16: for all d ∈ D do
17:     if release_cond(m) == true then
18:         forward(d);
19:         deferredList.delete(d);
20:     end if
21: end for
```

can be changed to hot as more measurements are collected in the system. Continuing with the above example, suppose node A later sees its estimation to B with newly collected information to be significantly different from its own past measurement. For example, for a new measurement of 10KB/s, node A may want to tell other nodes about the deferred experience (*release*).

Algorithm 2 illustrates details of the selective deferral and release technique. As in Algorithm 1, a node performs **initiate** when it obtains a new measurement, while non-source nodes perform **receive** when they receive dissemination messages from neighbors. If the measured information is hot to the system (i.e., *deferral_cond(m) == false*), it is immediately disseminated; otherwise, it is put in the deferred list, as seen in **initiate**. As before, these functions can be defined on the local node state. Any receiving node stores new information and simply forwards it if it has not seen the information before, as shown in the **receive** function. In both **initiate** and **receive**, a release test follows after new information is forwarded. This checks whether any prior deferred information is now hot and can be distributed, as shown in **release_test**. Although not shown explicitly in the algorithm, deferred messages will be purged, based on their age.

In this work, we define a deferral condition and a release condition based on the difference between new measurement and current estimation derived from prior measurements. Again, defining these conditions is application specific. To define a deferral condition, let us

suppose that $observed$ is a newly measured throughput to a specific server, and $expected$ is the estimated throughput to that server based on past measurements. The deferral condition is then defined as follows:

Deferral condition:

$$\frac{|observed - expected|}{observed} < \tau_1$$

If this condition is true, we defer dissemination for the given information. Thus, $\tau_1 = 0$ means no information will be deferred, whereas any arbitrary large value of τ_1 (e.g., $\tau_1 = 100$) may defer most of the newly collected measurements.

The release condition is similarly defined with the deferral condition by comparing deferred measurement ($deferred$) and current estimation ($expected$), as follows:

Release condition:

$$\frac{|deferred - expected|}{deferred} \geq \tau_2$$

Since $expected$ is the estimated throughput with all past relevant measurements, it can be different from the estimated value computed in the deferral phase. By this condition, if the deferred measurement has distinct information from the current estimation, it begins to be disseminated.

Defining τ-values may depend on application or system requirements. In Section 5.3, we examine how τ-values impact performance and dissemination overhead.

5 Evaluation

We now present results from an evaluation of OPEN using a trace-based simulation. We first describe our simulation methodology, followed by performance results for the two selection problems, i.e., resource and replica selection. Then, we examine the dissemination overhead of OPEN.

5.1 Simulation setup and methodology

We collected download traces for a span of 10 months (July 2007—April 2008) with 242 PlanetLab [36] nodes dispersed world-wide. For this data collection, we first placed data files on randomly selected nodes, and then generated random queries for actual downloading.

Table 3. Trace data

Data size	# of traces	# of nodes	# of objects	Mean elapsed (s)	Mean RTT (ms)
1M	22567	153	72	13.7	103
2M	25957	230	82	22.4	117
4M	28018	166	106	39.9	102
8M	26237	159	85	67.4	101
16M	11795	128	102	164.2	98

Before beginning the queries, all-pair pings were performed 30 times for each pair of nodes and the smallest RTTs were recorded as the latency of the node pair. Finally a thousand random queries were generated and the downloading elapsed times were recorded. For each query, we recorded the data object, the client, the server, the latency, and the elapsed time for downloading. This data collection was repeated multiple times to collect more records. Based on the analysis of GridFTP traces [26], we used 5 data sizes from 1 MB to 16 MB in our experiments. Table 3 provides the details of the download traces.

For evaluation, we design and implement a round-based simulator which uses the download traces as input and executes a selection algorithm. Initially, the simulator constructs a network in which nodes are *randomly* connected to each other with a predefined neighbor size without any locality or topological considerations. To minimize error due to the construction, we repeated simulations at least 10 times and report the results with 95% confidence intervals as needed. After constructing the network, the simulator runs each selection algorithm and the results are compared. At each round, it constructs a *virtual trace* in which randomly selected records are virtually composed. More in detail, the simulator chooses pairs of compute and data nodes randomly according to the experimental parameters, i.e., replication factor and candidate size. Then the download time for each pair is recorded in the *virtual trace* for that round. The simulator then selects one pair based on each selection algorithm, and returns the corresponding download time in the virtual trace as the selection result.

We demonstrate evaluation results for three systems according to the scale, *Small* ($n = 242$), *Medium* ($n = 1210$), and *Large* ($n = 12100$), but focus more on the large-scale system. We scaled the simulated system by allocating multiple simulated nodes to the same trace data node. Candidate size (c) is the number of candidate nodes for resource selection, while replication factor (r) is the number of replicas holding a data object.

For evaluation, we compare our OPEN-estimation based selection (OPEN) with a diverse set of selection

techniques. These include *random selection (*RANDOM*)* that randomly selects a node, and *latency-based selection (*PROXIM*)* that finds a client-server pair with the smallest RTT. In addition, we consider selection based on several *pairwise estimation* techniques that use only firsthand measurements. These techniques include statistical mean, median, exponential smoothing, and last value; we choose the *best* one of this group and call it PAIRWISE. For example, if selection by median yielded the best result at a round, we take that result (by median) as PAIRWISE for that round. For all estimation techniques, we assume infinite window size, thus all the past measurements are used in estimation.

Unlike RANDOM and PROXIM, the other selection techniques can suffer from estimation failures due to shortage of relevant measurements[3]. To avoid meaningless estimation values from impacting the selection algorithm, we use the PAIRWISE and OPEN estimation techniques only if at least the *half* the measurements required for estimation are available, based on our observation that performance gets degraded if we perform selection with less than half; otherwise we assume that the selection using these techniques falls back on latency-based selection.

To compare the different selection algorithms, we mainly use the metric *Optimality Ratio* (O.R.) [33], where optimal is an oracle-based algorithm. Thus, O.R.=1 means the selection technique chooses optimal. Since we used *mixed* data sets in simulation as mentioned, relative comparison is also more meaningful than providing absolute download times. We also examine *overhead* of dissemination. For this, we evaluate *number of messages* generated for dissemination of measurements to share in the system. The *normalized* number of messages refers to the average number of messages per round at each node.

5.2 Selection performance

We present our trace-based experimental results with respect to the two selection problems described earlier: replica selection to choose one of replicated servers and resource selection to choose a computational resource for task allocation. In this section, we assume complete sharing of secondhand measurement information to show the benefit of this information. In the following section we will discuss the overheads of sharing and how these costs can be reduced.

[3]PROXIM does not fail since the trace data includes latency information.

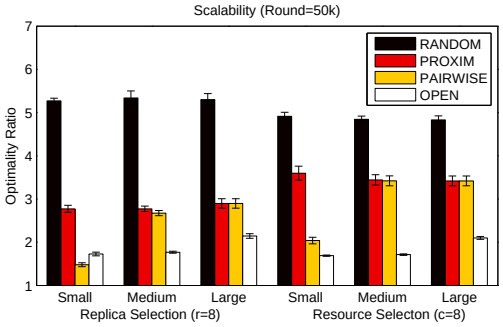


Figure 5. Performance comparison

5.2.1 Performance comparison

We begin by demonstrating the performance results in three different scaled systems. Figure 5 shows the results for both replica selection ($r = 8$) and resource selections ($c = 8$). In the small-sized system, PAIRWISE outperforms PROXIM, agreeing with the intuition that considering network throughput works better than relying on latency information for bandwidth-demanding applications. Similarly, OPEN significantly outperforms PROXIM by utilizing past measurements accrued in the system. We can see that OPEN and PAIRWISE are fairly comparable in the *Small* system.

However, PAIRWISE significantly degrades as the system scales up, yielding nearly equivalent results to PROXIM. This is because, as discussed in 3.1, there is a high probability that the pairwise techniques fail to see relevant measurements in their estimations, and hence will fall back to PROXIM. In replica selection, the fallback ratio to PROXIM is 15% in the *Small* system, but it increases 95% in the *Medium* system. In the *Large* system, it becomes almost 100%, indicating that no pairwise estimation was made due to lack of pair-level measurements. In contrast, OPEN falls back to PROXIM 0.5% in the *Small* system, 2% in the *Medium* system, and 18% in the *Large* system. *This result emphasizes again why secondhand estimation is attractive for large-scale systems.*

The fallback ratios for PAIRWISE are greater in resource selection; it is 24% in the Small system, while it is 15% for replica selection, and this explains why PAIRWISE shows better performance in replica selection in the *Small* system in Figure 5. Unlike this, OPEN shows similar fallback ratios between two selections. In the *Large* system, OPEN requires more rounds to collect measurements for each server.This slightly increases O.R. than the smaller systems in the *Large* system.

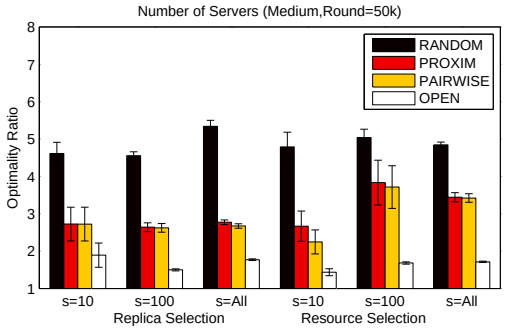


Figure 6. Impact of the number of servers

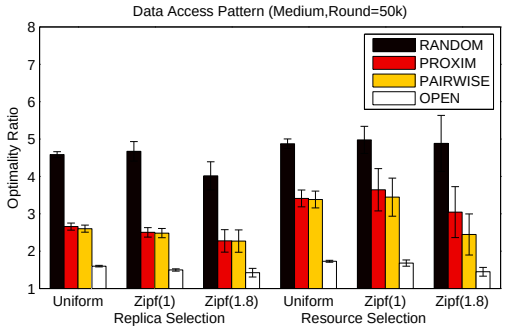


Figure 7. Impact of the data access patterns

Although OPEN shows good performance compared to other techniques, the results might depend on environmental factors. To examine this, we next perform experiments with different number of servers and data access patterns.

5.2.2 Impact of the number of servers

We next study the impact of the number of servers (s) on the performance of the different techniques. Intuitively, having fewer servers is likely to help PAIRWISE, as there would be greater likelihood of pairwise measurements to these servers being available. In this experiment, we set up three configurations: a small dedicated server environment ($s = 10$), a peer-to-peer computing environment in which any node can work as a server ($s = All$), and a medium in which roughly 10% of nodes work as data servers ($s = 100$). Since we observed PAIRWISE does not make any difference from PROXIM in the large-scale setting, we perform this experiment in the *Medium* system to closely see the impact of the number of servers.

Figure 6 shows performance results. With small, dedicated servers (i.e., $s = 10$ and $s = 100$), we can see a high degree of variations in both replica and resource selections. This is because the results should depend on the chosen servers more in the relatively small number of server environments. Despite the variations, the result shows that OPEN outperforms all the other techniques in diverse environments with different number of servers.

5.2.3 Impact of data access patterns

We next investigate the impact of data access pattern. Up to now, we have assumed uniform data access. In reality, however, data access distribution can be skewed, thus some objects can be more frequently retrieved [6, 14, 30, 35] showing *Zipf-like* distribution in which access frequency of ith-most popular object is proportional to $i^{-\alpha}$, where α is the Zipf parameter determining skewness.

Figure 7 demonstrates performance results under three different access patterns: uniform, Zipf with $\alpha = 1.0$ to emulate the Internet web request pattern [6], and Zipf with $\alpha = 1.8$ for an extremely skewed access pattern observed in [35]. As in the number of servers above, we perform this experiment in the *Medium* system. We can see relatively high variations for greater values of Zipf parameters. This is because a small set of servers could be more repeatedly accessed under the Zipf accesses, and performance would largely depend on the set of selected servers. Compared to this, OPEN shows more stable results, less affected by the data access patterns.

5.2.4 Impact of replication and candidate set size

We next consider its performance with respect to replication factor r and candidate set size c. For replica selection, we consider a highly replicated environment with $r = 32$ in addition to our basic setting $r = 8$. For resource selection, we consider more complex settings with candidate size and replication factor. We test following three settings: $c = 8$ and $r = 1$ (i.e., no replication), $c = 32$ and $r = 1$ (i.e., a greater set of candidates), and $c = 8, r = 8$ (i.e., resource selection in a replicated environment). Figure 8 shows that OPEN is superior to the other techniques under the diverse settings. In addition to O.R., Table 4 shows *mean downloading time* in milliseconds and how much OPEN saves compared to PROXIM. This includes the learning phase where OPEN relies on latency information. When we compute mean elapsed time after excluding this learning phase performance, we obtain 35% saving on average compared to

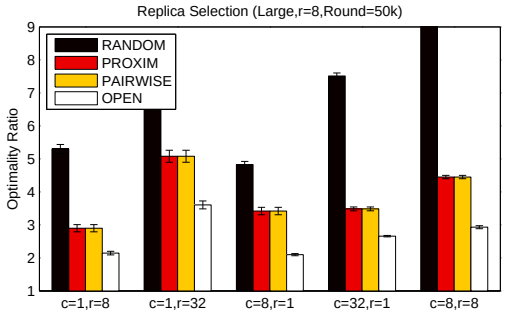


Figure 8. Impact of replication and candidate size

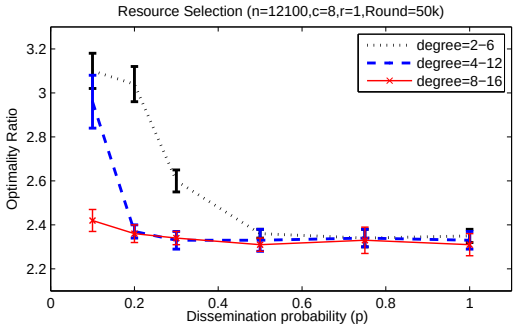


Figure 9. Impact of system degree

Table 4. Mean downloading time

Replication factor	Candidate size	PROXIM (msec)	OPEN (msec)	Saving
8	1	44112	33586	24%
32	1	39359	30966	21%
1	8	62653	43816	30%
1	32	45842	34410	25%
8	8	27459	17519	36%

27% saving including the learning phase performance. In the table, the 95% confidence intervals are smaller than 1500 milliseconds.

We showed OPEN consistently outperforms the other techniques in diverse settings as well as various working environments. We next examine the overhead of the OPEN framework, and discuss how to handle the dissemination overhead with minimal performance loss.

5.3 Overhead

OPEN needs to share measurements among nodes for secondhand estimation. In the previous section, we simply assumed all the past measurements are available to all the nodes by flooding of measurements. In this section, we show how we can optimize the cost of measurement sharing but with no significant performance loss.

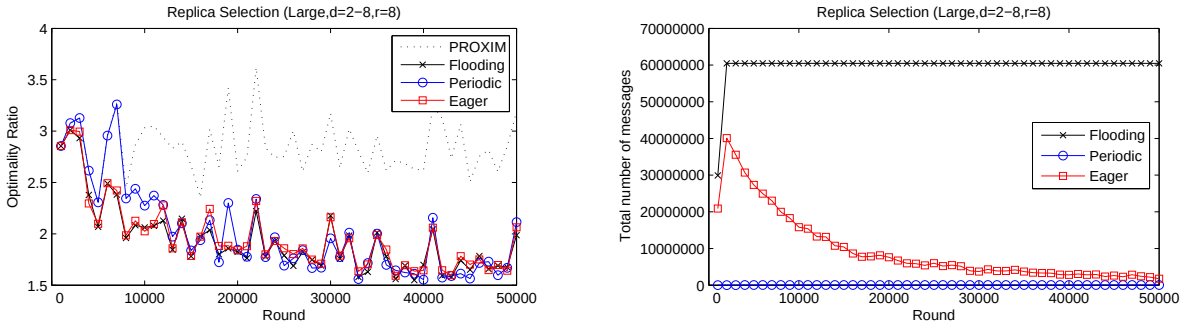
5.3.1 Impact of system degree

We first investigate the impact of system degree (i.e., the average neighbor size in the system) over dissemination probability. Figure 9 shows performance results for resource selection over dissemination probability. In the figure, we can clearly see different convergence according to system degree. As discussed in 4.2.1, systems with different degrees respond differently to dissemination probability. The high-degree system (d=8–16) converges quickly even with very small probability ($p \approx 0.1$). In contrast, the small-degree system (d=2–6) converges with $p \approx 0.5$. We observed that these probabilities result in around 70–80% successful dissemination rates (rather than perfect dissemination). However, there could be other chances to obtain measurements headed to the same server despite losing the current one.

5.3.2 Selective eager dissemination

We continue to examine performance and overhead of selective eager dissemination. With this technique, critical information is distributed without any delay, while non-critical information relies on periodic, lazy dissemination. Recall that the decision whether new measurement is critical is based on a counter that how many measurements have been propagated for *that* server. If the counter value is below the predetermined threshold, the measurement will eagerly distributed; otherwise, it should wait until next periodic timer goes off.

For evaluation, we examine three different settings: *Flooding*, *Periodic* with interval 1000, and *Eager* with the same interval but with an eager threshold 2. In other words, with *Eager*, any new measurements will be eagerly forwarded if the node has seen less than 2 measurements for the corresponding server, while others will be periodically disseminated with the interval. Figure 10 presents experimental results in replica selection with replication $r = 8$ in the *Large* system. We assumed node degree is uniformly distributed between 2 and 8. In Figure 10(a), we can see that *Periodic* suffers from performance degradation due to the large dissemination interval at the first stage of the time, showing almost similar with PROXIM. In contrast, *Eager* yields comparable performance to *Flooding* even from the first stage. Figure 10(b) shows the number of messages to disseminate measurements for each round. At the first stage, *Eager* creates a large number of dissemination messages, but it



(a) Performance

(b) Overhead

Figure 10. Selective eager dissemination

is significantly reduced over time, approaching *Periodic*.

In the above experiment, the total number of dissemination messages for *Eager* was $\sim 15\%$ of the *Flooding* result. This can be further optimized by taking advantage of dissemination probability. Figure 11 shows experimental results for selective eager dissemination with diverse dissemination probabilities. In this experiment, we gave the same interval and threshold as the above but each *Eager* has different dissemination probability; for example, *Eager(0.1)* stands for selective eager dissemination with probability 0.1. As seen in Figure 11(a), a small dissemination probability fails sharing measurements; however, proper optimization, for example *Eager(0.3)*, yields fairly comparable results to flooding and *Eager(1.0)*. In this case, the overhead is further reduced to $\sim 30\%$ of the *Eager(1.0)* result, equivalent to only $\sim 5\%$ of *Flooding*.

5.3.3 Selective deferral and release

We next evaluate the selective deferral and release technique. Table 5 presents experimental results in replica selection with replication $r = 8$ in the *Large* system with the same node degree distribution as in the above section 5.3.2. In this experiment, we use $\tau = \tau_1 = \tau_2$ to make deferral and release decisions. Smaller τ would make deferral decision less likely, whereas greater τ tends to aggressively defer dissemination of new measurements. As shown in the table, we can see that performance degrades with a greater τ value due to the increasing number of deferred measurements. The number of released measurements increases until $\tau = 0.25$, but decreases as τ becomes greater. This is because with a smaller τ value than 0.25, a large portion of measurements are propagated without having been deferred, thus have a smaller chance of being releasing later; however,

Table 5. Impact of selective deferral and release

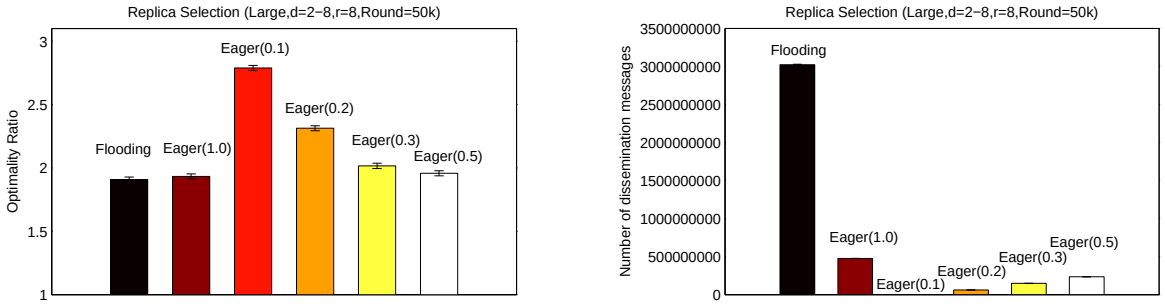
τ	O.R.	# Deferred	# Released	Saving
0	2.09	0	0	0%
0.1	2.09	9359	4876	9%
0.25	2.10	21266	7204	28%
0.5	2.14	32846	4903	56%
1	2.25	38875	488	77%
100	2.32	44573	0	89%

with greater τ, release condition becomes more strict with a greater τ value, thus suppressing release of the deferred measurements. The table shows a trade-off between performance and overhead, suggesting a sweet spot lies somewhere between $\tau = 0.25$ and $\tau = 0.5$.

Figure 12 plots the number of deferred and released measurements over time for $\tau = 0.25$ and $\tau = 0.5$. Interestingly, rate of message growth for deferred measurements are greater than for released measurements, implying that saving will be greater over time. The number of messages for deferred measurements are much more sensitive to τ then for release. As seen in Table 5, however, 34% of deferred measurements are released for $\tau = 0.25$, while only 15% were released for $\tau = 0.5$ with a tighter release condition.

5.4 Simulation with S^3 Data Sets

To test the generality of the framework, we conducted another simulation with different data sets created by another institution. The HP S^3 project [19, 48] measures end-to-end bandwidth, including capacity, available bandwidth, and loss rate, for all pairs in PlanetLab. We used S^3 bandwidth measurements released on October 12, 2009. Since the S^3 project still does not pro-



(a) Performance

(b) Overhead

Figure 11. Selective eager dissemination with dissemination probability

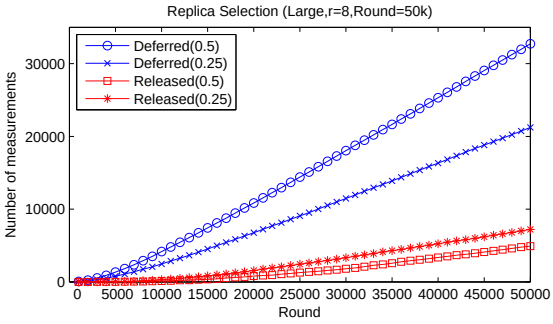


Figure 12. Number of deferred and released measurements

Table 6. Comparison of data sets

	Our Data Set	S^3 Data Set
Number of nodes	242	373
Number of clients	238	250
Number of servers	183	367
Number of pairs	17,296	78,693
Number of measurements	114,574	78,693

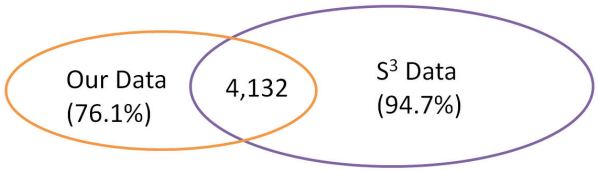


Figure 13. Pair distribution diagram for S^3 and our data sets

vide end-to-end latency information, we used the PlanetLab all-pair ping data set measured on September 24, 2009 [18, 40].

Table 6 compares two data sets between our collection and the S^3 measurements. In addition, Figure 13 illustrates what proportion of PlanetLab node pairs are overlapped in both data sets. As shown in the figure, over 94% of pairs in the S^3 data set are not included in our data collection. Similarly, 76% of pairs in our data set are not included in the S^3 data set. Thus, simulation with the S^3 measurements would be helpful for verification of generality for the OPEN framework.

Figure 14 shows selection performance for both replica selection and resource selection. We can see that OPEN outperforms other techniques in any system size. In replica selection, PROXIM yields O.R. $\approx$ 2.8; in contrast, we can see that OPEN shows O.R $<$ 2.0 in *Small* and *Medium* systems. As discussed in Section 5.2.1, OPEN shows a little greater ratio in the *Large* system, which needs many more rounds to distribute measurements than the smaller systems. We can see that PAIRWISE does not have benefits due to a high degree of fallback ratio.

The results in resource selection are more dramatic: PROXIM degrades to O.R $\approx$ 6.0, while OPEN shows O.R $\approx$ 3.0, even in the *Large* system, as shown in the figure. We presume that there is a greater degree of node heterogeneity, particularly with respect to networking capability, with many more nodes in the S^3 data set. In replica selection, this kind of heterogeneity is not critical because the compute node is fixed (in other words, we do not choose a compute node in replica selection); in contrast, it may be critical in resource selection because node heterogeneity could significantly affect the downloading performance.

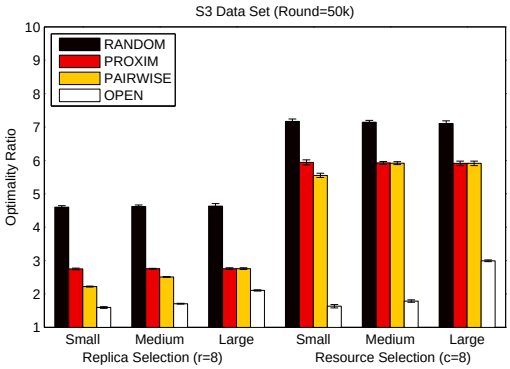


Figure 14. Performance comparison with S^3 data set

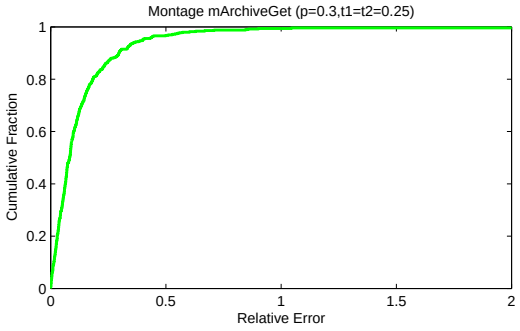


Figure 15. Relative error of OPEN estimates (Montage)

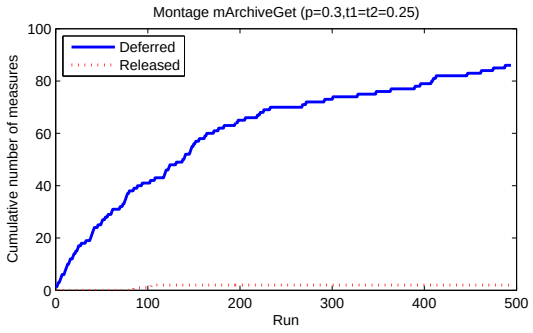


Figure 16. Number of deferral/release measures (Montage)

5.5 Running Montage on the OPEN framework

We launched our OPEN framework in PlanetLab with 50 nodes. We constructed an overlay network by using FreePastry [12]. Each node maintains a local measurement table and an imported measurement table for sharing measurements. The neighbor size we used is 8, and we set the dissemination probability to 0.3 (the same value we used in the paper). We also configured the selective deferral and release with a deferral parameter of $\tau_1 = 0.25$ and a release parameter of $\tau_2 = 0.25$. With this setting, if the measured value is located between (*0.75*expected*) and (*1.25*expected*), it will be deferred in dissemination; otherwise, it will be disseminated to its neighbor nodes, based on the dissemination probability.

Montage is a toolkit for astronomical research, which enables astronomers to conduct a variety of domain-specific experiments. In particular, Montage provides the functionality to retrieve space images formatted by FITS (Flexible Image Transport System), a standard format for image representation in astronomy, and for combining these images to construct universe mosaics. We launched a Montage application in our OPEN framework. The application accesses FITS images from a remote Montage server by using a Montage tool `mArchiveGet`, a retrieval tool for a FITS image, based on the given URL (http://archive.stsci.edu/). The number of FITS we retrieved is 36 images, the size of which is ranged from 6MB to 9MB.

We created a total of 537 queries, and the framework could make 494 estimations, except for initial learning. In the beginning of the experiment, we collected latency between the PlanetLab nodes and the Montage server for the distance metric in estimation. Figure 15 shows a cumulative distribution of the relative error between the collected measures and the corresponding estimates. We can see a high degree of estimation accuracy even with secondhand measures showing that 95% of the estimations are located in 0.4 relative error, despite geographical differences and heterogeneity of the PlanetLab nodes.

In this experiment, we observed 86 deferred messages and 2 released messages, indicating 17% additional saving over the saving by probabilistic dissemination, even with a tight deferral condition ($\tau_1 = 0.25$). Figure 16 presents the cumulative number of deferral and released messages over time. We can see that new measures are deferred in dissemination, and many of them are not released.

The second experiment with Montage is resource selection with 3 selection techniques: random selection (RANDOM), latency-based selection (PROXIM), and OPEN-based (OPEN). In this experiment, we considered two candidate sizes 8 and 16. For each Montage query,

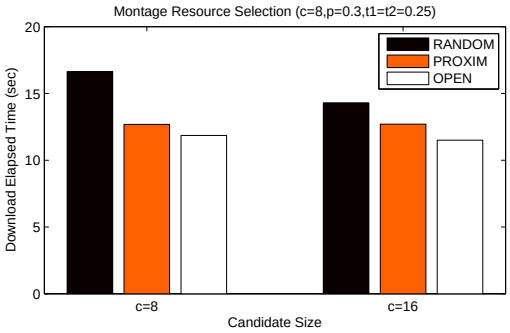


Figure 17. Resource selection performance (Montage)

we randomly constructed a candidate set, based on the candidate size. Then, we choose one candidate for each selection technique. The chosen node performed the Montage query, and the downloading elapsed time was recorded. If any query failed in the interleaved set of queries, the result was discarded in our analysis. We present the average download elapsed times of the selection results where OPEN and PROXIM made different decisions.

Figure 17 shows the downloading elapsed times for selection techniques. For candidate size 8 ($c = 8$), the total number of queries is 1,600, and OPEN made 169 different selections from PROXIM, while these numbers were 2,918 and 200, respectively, for candidate size 16 ($c = 16$). The results shown in the figure confirm that OPEN outperforms existing selection techniques in a live setting with a real application, as well. In the figure, OPEN yields a greater gap from PROXIM with the bigger candidate set.

5.6 Discussion

An important question is whether the overheads of dissemination might swamp the gains. Although random selection yielded poor and unstable results, it did not create any additional cost for the purpose of estimation. However, any selection based on estimations would incur extra load and traffic which may affect user data access. For example, for selective deferral and release with $p = 0.3$ and $\tau = 0.25$, we observed that each node created 1.15 MB additional traffic on average to share 50,000 measurements representing 50,000 distinct downloads over time[4]. In the same setting, Spruce [43] requires 3.6 GB traffic per node (based on 300 KB per measurement that is the traffic requirement on average in Spruce). Given the rich availability of peer-to-peer bandwidth, and the time-frame for sharing 50,000 distinct downloads, the OPEN overhead is likely to have minor impact on the results. In addition, dissemination messages can be piggybacked over other system messages to reduce the number of extra messages, e.g., periodic neighbor heartbeats needed for system health.

Another issue would be "information inequality" due to different joining times or imperfect probabilistic dissemination. This may result in different decisions even for the same event at each node. In the selective eager dissemination technique, each node makes its own eager or periodic forwarding decision. Similarly, in the selective deferral and release technique, the source node makes a decision whether new information is distributed immediately or not. Those decisions rely on local information, and thus can be biased. For example, source node S is long-lived and can make a deferral decision because it has redundant information, but any recently joined node may suffer from estimation failure due to lack of relevant information which should have been available if S released it. This information inequality can be mitigated by downloading shared measurements from parent nodes at joining times.

6 Related Work

A system with similar goals to ours is NWS [46, 47]. NWS maintains network sensors measuring network performance by periodic probing and statistical predictions based on probing results. Thus it requires $O(n^2)$ communications for all-pair probing, where n is the number of sensors. To reduce the probing overhead, the sensors are organized in a hierarchical structure by configuring logical clusters called cliques, and end-to-end network performance between two nodes lying different clusters is determined by measured network performance between the sensors in those clusters. While accurate for sensor nodes, other nodes estimation may be inaccurate in non-clustered distributed environments. In addition, all-pair probing is still expensive even with a hierarchical design, particularly in large-scale systems this paper considers. OPEN enables scalable, low-cost estimation by sharing secondhand measurements, i.e., without relying on all-pair probing.

iPlane [31] is an infrastructure-based service which provides prediction of end-to-end network performance in the Internet. For the prediction service, iPlane measures segment paths chosen based on the Internet topol-

[4]We consider 40 bytes for one dissemination message including TCP header based on Table 2.

ogy, and the end-to-end path property is inferred from the collected measurements. While providing fairly accurate estimation, it requires explicit probing between additionally deployed probing sensors.

Gossip-based dissemination is scalable and resilient to failure, so it is used in many areas, such as wireless ad hoc networks [16, 27] and large-scale networks [11, 15, 20, 21, 45], for diverse purposes like routing, data dissemination, membership management, and so forth. Haas et al. [16] proposed a set of gossip protocols with several parameters, including forwarding probability, number of hops, and neighbor size, to improve ad hoc routing protocols, many of which are based on flooding. Although gossip techniques can significantly reduce the dissemination overhead with negligible information loss, we consider application-level semantics with respect to information criticality for further reduction of dissemination overhead in this paper.

GridFTP [1] is a file transfer tool widely used in the grid community. Kourtellis et al. [26] performed a workload analysis study with GridFTP traces collected from all over the world for about 1.5 years. According to their results, 80% of transferred data were less than 16 MB and 94% of the total were less than 32 MB. They also observed that over 70% of the total volume were contributed by inter-organization transfers. Our simulation traces were selected based on the observations.

7 Conclusion

Large-scale distributed systems are attractive for many distributed computing applications by providing a scalable, cost-effective infrastructure. However, a primary challenge is the unpredictability of data access which can lead to significant performance bottlenecks. Providing predictability in data access is needed to accommodate the large set of data-intensive computing applications.

To this end, we have designed a framework called OPEN which offers end-to-end network performance estimation based on secondhand measurements observed other nodes in the system. To share secondhand measurements, OPEN proactively distributes newly collected measurements by a probabilistic dissemination technique. For evaluation, we conducted extensive experiments with our PlanetLab trace data sets and a Montage application in a live setting, and the experimental results show that resource and replica selections with OPEN consistently outperform selection techniques based on statistical pairwise estimations as well as latency-based selection. In addition, OPEN can dramatically reduce dissemination overhead to share secondhand measurements without any significant performance loss by several optimization techniques such as selective eager dissemination and selective deferral and release of new measurements.

OPEN will ideally fit large-scale desktop grids harnessing idle cycles. We further believe that our framework can be applied to existing grid systems having a multi-site, cluster-based architecture, since we made no assumptions about topological constraints. For example, secondhand measurements from nodes in the same cluster (or virtual organization) can be given more weight in the estimation process. We plan to optimize OPEN to better support such cluster-based, large-scale systems as our future work.

References

[1] William Allcock, John Bresnahan, Rajkumar Kettimuthu, and Michael Link. The globus striped gridftp framework and server. In *Proceedings of ACM/IEEE Conference on Supercomputing (SC '05)*, 2005.

[2] David P. Anderson and Gilles Fedak. The computational and storage potential of volunteer computing. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '06)*, pages 73–80, 2006.

[3] Matthew Andrews, Bruce Shepherd, Aravind Srinivasan, Peter Winkler, and Francis Zane. Clustering and server selection using passive monitoring. In *Proceedings of INFOCOM (INFOCOM '02)*, pages 1717–1725, 2002.

[4] 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.

[5] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, and Mei-Hui Su. Characterization of scientific workflows. In *Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS08)*, 2008.

[6] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of INFOCOM (INFOCOM '99)*, pages 126–134, 1999.

[7] Michael Cardosa and Abhishek Chandra. Resource bundles: Using aggregation for statistical wide-area resource discovery and allocation. In *Proceedings of 28th International Conference on Distributed Computing Systems (ICDCS '08)*, pages 760–768, 2008.

[8] Andrew Chien, Brad Calder, Stephen Elbert, and Karan Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5):597–610, 2003.

[9] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *Proceedings of ACM SIGCOMM (SIGCOMM '04)*, pages 15–26, 2004.

[10] Ewa Deelman and Ann Chervenak. Data management challenges of data-intensive scientific workflows. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)*, pages 687–692, 2008.

[11] Mayur Deshpande, Bo Xing, Iosif Lazardis, Bijit Hore, Nalini Venkatasubramanian, and Sharad Mehrotra. Crew: A gossip-based flash-dissemination system. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS '06)*, page 45, 2006.

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

[13] The Globus Alliance, http://www.globus.org/.

[14] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. *SIGOPS Operating Systems Review*, 37(5):314–329, 2003.

[15] Indranil Gupta, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE Transactions on Parallel and Distributed Systems*, 17(7), 2006.

[16] Zygmunt J. Haas, Joseph Y. Halpern, and Li Li. Gossip-based ad hoc routing. *IEEE/ACM Transactions on Networking*, 14(3):479–491, 2006.

[17] Qi He, Constantine Dovrolis, and Mostafa Ammar. On the predictability of large transfer tcp throughput. In *Proceedings of ACM SIGCOMM (SIGCOMM '05)*, pages 145–156, 2005.

[18] Sing Wang Ho, Thom Haddow, Jonathan Ledlie, Moez Draief, and Peter Pietzuch. Deconstructing internet paths: An approach for as-level detour route discovery. In *Proceedings of the 8th International Workshop on Peer-to-Peer Systems (IPTPS '09)*, 2009.

[19] Scalable sensing service (s^3): http://networking.hpl.hp.com/s-cube/pl/.

[20] Meng jang Lin and Keith Marzullo. Directional gossip: Gossip in a wide area network. In *Proceedings of European Dependable Computing Conference*, pages 364–379, 1999.

[21] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, 2003.

[22] Jik-Soo Kim, Beomseok Nam, Peter Keleher, Michael Marsh, Bobby Bhattacharjee, and Alan Sussman. Resource discovery techniques in distributed desktop grid environments. In *Proceedings of GRID (GRID '06)*, September 2006.

[23] Jinoh Kim, Abhishek Chandra, and Jon B. Weissman. Accessibility-based resource selection in loosely-coupled distributed systems. In *Proceedings of 28th International Conference on Distributed Computing Systems (ICDCS '08)*, pages 777–784, 2008.

[24] Jinoh Kim, Abhishek Chandra, and Jon B. Weissman. Using data accessibility for resource selection in large-scale distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):788–801, 2009.

[25] Derrick Kondo, Andrew A. Chien, and Henri Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing (SC '04)*, 2004.

[26] Nicolas Kourtellis, Lydia Prieto, Adriana Iamnitchi, Gustavo Zarrate, and Dan Fraser. Data transfers in the grid: workload analysis of globus gridftp. In *Proceedings of the 2008 international workshop on Data-aware distributed computing (DADC '08)*, pages 29–38, 2008.

[27] Pradeep Kyasanur, Romit Choudhury, and Indranil Gupta. Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, 0:91–100, 2006.

[28] LHC: Large hadron collider, http://lhc.web.cern.ch/lhc/.

[29] Virginia Lo, Daniel Zappala, Dayi Zhou, Yuhong Liu, and Shanyu Zhao. Cluster computing on the fly: P2p scheduling of idle cycles in the internet. In *Proceedings of the IEEE Fourth International Conference on Peer-to-Peer Systems*, pages 227–236, 2004.

[30] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of ACM SIGMETRICS (SIGMETRICS '02)*, pages 258–259, 2002.

[31] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An information plane for distributed services. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI '06)*, 2006.

[32] E. Ng and H. Zhang. Predicting internet network distance with coordiantes-based approaches. In *Proceedings of IEEE INFOCOM (INFOCOM '02)*, pages 170–179, 2002.

[33] T. S. Eugene Ng, Yang hua Chu, Sanjay G. Rao, Kunwadee Sripanidkulchai, and Hui Zhang. Measurement-based optimization techniques for bandwidth-demanding peer-to-peer systems. In *Proceedings of INFOCOM (INFOCOM '03)*, pages 2199–2209, 2003.

[34] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *Proceedings of Proceedings of ACM High Performance Distributed Computing (HPDC '05)*, 2005.

[35] Venkata N. Padmanabhan and Lili Qiu. The content and access dynamics of a busy web site: findings and implications. *SIGCOMM Comput. Commun. Rev.*, 30(4):111–123, 2000.

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

[37] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of Proceedings of ACM High Performance Distributed Computing (HPDC '98)*, page 140, 1998.

[38] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[39] S. Seshan, M. Stemm, and R. H Katz. SPAND: Shared Passive Network Performance Discovery. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 135–146, Monterey, CA, December 1997.

[40] http://www.iis.ee.imperial.ac.uk/ singwang/.

[41] Sloan digital sky survey / skyserver, http://cas.sdss.org/dr7/en/.

[42] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM (SIGCOMM '01)*, pages 149–160, 2001.

[43] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC '03)*, pages 39–44, 2003.

[44] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[45] Spyros Voulgaris and Maarten van Steen. Hybrid dissemination: adding determinism to probabilistic multicasting in large-scale p2p systems. pages 389–409, 2007.

[46] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15:757–768, 1999.

[47] Rich Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS Performance Evaluation Reviews*, 30(4):41–49, 2003.

[48] Praveen Yalagandula, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Sung-Ju Lee. S3: a scalable sensing service for monitoring large networked systems. In *Proceedings of the 2006 SIGCOMM workshop on Internet network management (INM '06)*, pages 71–76, 2006.

[49] Rongmei Zhang, Chunqiang Tang, Y. Charlie Hu, Sonia Fahmy, and Xiaojun Lin. Impact of the inaccuracy of distance prediction algorithms on internet applications - an analytical and comparative study. In *Proceedings of INFOCOM (INFOCOM '06)*, 2006.

[50] D. Zhou and V. Lo. Cluster computing on the fly: resource discovery in a cycle sharing peer-to-peer system. In *Proceedings of the 2004 Fourth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '04)*, pages 66–73, 2004.