# Service Placement in a Shared Wide-Area Platform

David Oppenheimer[1], Brent Chun[2], David Patterson[3], Alex C. Snoeren[1], and Amin Vahdat[1]
[1]*UC San Diego,* [2]*Arched Rock Corporation,* [3]*UC Berkeley*
{*doppenhe,snoeren,vahdat*}*@cs.ucsd.edu, bnc@theether.org, pattrsn@cs.berkeley.edu*

## ABSTRACT

Emerging federated computing environments offer attractive platforms to test and deploy global-scale distributed applications. When nodes in these platforms are time-shared among competing applications, available resources vary across nodes and over time. Thus, one open architectural question in such systems is how to map applications to available nodes—that is, how to discover and select resources. Using a six-month trace of PlanetLab resource utilization data and of resource demands from three long-running PlanetLab services, we quantitatively characterize resource availability and application usage behavior across nodes and over time, and investigate the potential to mitigate the application impact of resource variability through intelligent service placement and migration.

We find that usage of CPU and network resources is heavy and highly variable. We argue that this variability calls for intelligently mapping applications to available nodes. Further, we find that node placement decisions can become ill-suited after about 30 minutes, suggesting that some applications can benefit from migration at that timescale, and that placement and migration decisions can be safely based on data collected at roughly that timescale. We find that inter-node latency is stable and is a good predictor of available bandwidth; this observation argues for collecting latency data at relatively coarse timescales and bandwidth data at even coarser timescales, using the former to predict the latter between measurements. Finally, we find that although the utilization of a particular resource on a particular node is a good predictor of that node's utilization of that resource in the near future, there do not exist correlations to support predicting one resource's availability based on availability of other resources on the same node at the same time, on availability of the same resource on other nodes at the same site, or on time-series forecasts that assume a daily or weekly regression to the mean.

## 1. INTRODUCTION

Federated geographically-distributed computing platforms such as PlanetLab [3] and the Grid [7, 8] have recently become popular for evaluating and deploying network services and scientific computations. As the size, reach, and user population of such infrastructures grow, resource discovery and resource selection become increasingly important. Although a number of resource discovery and allocation services have been built [1, 11, 15, 22, 28, 33], there is little data on the utilization of the distributed computing platforms they target. Yet the design and efficacy of such services depends on the characteristics of the target platform. For example, if resources are typically plentiful, then there is less need for sophisticated allocation mechanisms. Similarly, if resource availability and demands are predictable and stable, there is little need for aggressive monitoring.

To inform the design and implementation of emerging resource discovery and allocation systems, we examine the usage characteristics of PlanetLab, a federated, time-shared platform for "developing, deploying, and accessing" wide-area distributed applications [3]. In particular, we investigate variability of available host resources across nodes and over time, how that variability interacts with resource demand of several popular long-running services, and how careful application placement and migration might reduce the impact of this variability. We also investigate the feasibility of using stale or predicted measurements to reduce overhead in a system that automates service placement and migration.

Our study analyzes a six-month trace of node, network, and application-level measurements. In addition to presenting a detailed characterization of application resource demand and free and committed node resources over this time period, we analyze this trace to address the following questions: (i) Could informed service placement—that is, using live platform utilization data to choose where to deploy an application—outperform a random placement? (ii) Could migration—that is, moving deployed application instances to different nodes in response to changes in resource availability—potentially benefit some applications? (iii) Could we reduce the overhead of a service placement service by using stale or predicted data to make placement and migration decisions? We find:

- CPU and network resource usage are heavy and highly variable, suggesting that shared infrastructures such as PlanetLab would benefit from a resource allocation infrastructure. Moreover, available resources across nodes and resource demands across instances of an application both vary widely. This suggests that even in the absence of a resource allocation system, some applications could benefit from intel-

ligently mapping application instances to available nodes.

- Node placement decisions can become ill-suited after about 30 minutes, suggesting that a resource discovery system should not only be able to deploy applications intelligently, but should also support migrating performance-sensitive applications whose migration cost is acceptable.
- Stale data, and certain types of predicted data, can be used effectively to reduce measurement overhead. For example, using resource availability and utilization data up to 30 minutes old to make migration decisions still enables our studied applications' resource needs to be met more frequently than not migrating at all; this suggests that a migration service for this workload need not support a high measurement update rate. In addition, we find that internode latency is both stable and a good predictor of available bandwidth; this observation argues for collecting latency data at relatively coarse timescales and bandwidth data at even coarser timescales, using the former to predict the latter between measurements.
- Significant variability in usage patterns across applications, combined with heavy sharing of nodes, precludes significant correlation between the availability of different resources on the same node or at the same site. For example, CPU availability does not correspond to memory or network availability on a particular node, or to CPU availability on other nodes at the same site. Hence, it is not possible to make accurate predictions based on correlations within a node or a site. Furthermore, because PlanetLab's user base is globally distributed and applications are deployed across a globally distributed set of nodes, we note an absence of the cyclic usage pattern typical of Internet services with geographically colocated user populations. As a result, it is not possible to make accurate resource availability or utilization predictions for this platform based on time-series forecasts that assume a daily or weekly regression to the mean.

The remainder of this paper is organized as follows. Section 2 describes our data sources and methodology. Section 3 surveys platform, node, and network resource utilization behavior; addresses the usefulness of informed service placement; and describes resource demand models for three long-running PlanetLab services—CoDeeN [26], Coral [10], and OpenDHT [20]—that we use there and in subsequent sections. Section 4 investigates the potential benefits of periodically migrating service instances. Section 5 analyzes the feasibility of making placement and migration decisions using stale or predicted values. Section 6 discusses related work, and Section 7 concludes.

## 2. DATA SOURCES AND METHODOLOGY

We begin by describing our measurement data sources and PlanetLab, our target computing infrastructure. We then describe our study's methodology and assumptions.

### 2.1 System environment

PlanetLab is a large-scale federated computing platform. It consists of over 500 nodes belonging to more than 150 organizations at over 250 sites in more than 30 countries. In general, approximately two-thirds of these nodes are functioning at any one time. PlanetLab currently performs no coordinated global scheduling, and users may deploy applications on any set of nodes at any time.

All PlanetLab nodes run identical versions of Linux on x86 CPUs. Applications on PlanetLab run in *slices*. A slice is a set of allocated resources distributed across platform nodes. From the perspective of an application deployer, a slice is roughly equivalent to a user in a traditional Unix system, but with additional resource isolation and virtualization [3], and with privileges on many nodes. The most common slice usage pattern is for an application deployer to run a single distributed application in a slice. This one-to-one correspondence between slices and applications is true for the applications we characterize in Section 3.2, and for many other slices as well. The set of allocated resources on a single node is referred to as a *sliver*. When we discuss "migrating a sliver," we mean migrating the process(es) running on behalf of one slice on one node, to another node. We study approximately six months of PlanetLab node and network resource utilization data, from August 12, 2004 to January 31, 2005, collected from the following data sources. All sources periodically archive their measurements to a central node.

*CoTop* [17] collects data every 5 minutes on each node. It collects node-level information about 1-, 5-, and 15-minute load average; free memory; and free swap space. Additionally, for each sliver, CoTop collects a number of resource usage statistics, including average send and receive network bandwidth over the past 1 and 15 minutes, and memory and CPU utilization. *All-pairs pings* [24] measures the latency between all pairs of PlanetLab nodes every 15 minutes using the Unix "ping" command. *iPerf* [5] measures the available bandwidth between every pair of PlanetLab nodes once to twice a week, using a bulk TCP transfer.

In this study, we focus on CPU utilization and network bandwidth utilization. With respect to memory, we observed that almost all PlanetLab nodes operated with their physical memory essentially fully committed on a contin-

uous basis, but with a very high fraction of their 1 GB of swap space free. In contrast, CPU and network utilization were highly variable.

In our simulations, we use CoTop's report of per-sliver network utilization as a measure of the sliver's network bandwidth demand, and we assume a per-node bandwidth capacity of 10 Mb/s, which was the default bandwidth limit on PlanetLab nodes during the measurement period. We use CoTop's report of per-sliver %CPU utilization as a proxy for the sliver's CPU demand. We use $\frac{1}{load+1} * 100\%$ to approximate the %CPU that would be available to a new application instance deployed on a node. Thus, for example, if we wish to deploy a sliver that needs 25% of the CPU cycles of a PlanetLab node and 1 Mb/s of bandwidth, we say that it will receive "sufficient" resources on any node with $load <= 3$ and on which competing applications are using at most a total of 9 Mb/s of bandwidth.

Note that this methodology of using observed resource utilization as a surrogate for demand tends to under-estimate true demand, due to resource competition. For example, TCP congestion control may limit a sliver's communication rate when a network link the sliver is using cannot satisfy the aggregate demand on that link. Likewise, when aggregate CPU demand on a node exceeds the CPU's capabilities, the node's process scheduler will limit each sliver's CPU usage to its "fair share." Obtaining true demand measurements would require running each application in isolation on a cluster, subjecting it to the workload it received during the time period we studied. Note also that the resource demand of a particular sliver of an isolated application may change once that application is subjected to resource competition, even if the workload remains the same, if that competition changes how the application distributes work among its slivers. (Only one of the applications we studied took node load into account when assigning work; see Section 3.2.)

Finally, recent work has shown that using $\frac{1}{load+1} * 100\%$ to estimate the %CPU available to a new process underestimates actual available %CPU, as measured by an application running just a spin-loop, by about 10 percentage points most of the time [23]. The difference is due to PlanetLab's proportional share node CPU scheduling policy, which ensures that all processes of a particular sliver together consume no more than $\frac{1}{m}$ of the CPU, where *m* is the number of slivers demanding the CPU at that time. Spin-loop CPU availability measurements are not available for the time period in our study.

## 2.2 Methodology

We take a two-pronged approach to answering the questions posed in Section 1. First we examine node-level measurement data about individual resources, independent of any application model. We then compose this data about available node resources with simple models of application resource demand derived from popular PlanetLab services. This second step allows us to evaluate how space-and-time-varying sliver resource demand interacts with space-and-time-varying free node resources to determine the potential benefits of informed sliver placement and migration, as well as the impact of optimizations to a migration service.

We ask our questions from the perspective of a single application at a time; that is, we assume all other applications behave as they did in the trace. Thus, our approach combines *measurement, modeling,* and *simulation*, but the simulation does not consider how other users might react to one user's placement and migration decisions. We leave to future work developing a multi-user model that might allow us to answer the question "if multiple PlanetLab users make placement and migration decisions according to a particular policy, how will the system evolve over time?"

# 3. DEPLOYMENT-TIME PLACEMENT

If the number of individuals wishing to deploy applications in a shared distributed platform is small, or the number of nodes each user needs is small, a platform architect might consider space-sharing nodes rather than time-sharing. Giving users their own dedicated set of nodes eliminates node resource competition and thereby removes the primary motivation of informed application placement—finding lightly utilized nodes and avoiding overutilized ones. Therefore, we first ask whether such a space-sharing scheme is practical, by investigating what portion of global resources application deployers typically wish to use.

Figure 1 shows the number of PlanetLab nodes used by each slice active during our measurement period. We say a slice is "using" a node if the slice has at least one task in the node's process table (i.e., the slice has processes on that node, but they may be running or sleeping at the time the measurement is taken). More than half of PlanetLab slices run on at least 50 nodes, with the top 20% operating on 250 or more nodes. This is not to say, of course, that all of those applications "need" that many nodes to function; for example, if a platform like PlanetLab charged users per node they use, we expect that this distribution would shift towards somewhat lower numbers of nodes per slice. However, we expect that a desire to maximize realism in experimental evaluation, to stress-test application scalability, to share load among nodes for improved performance, and/or to maximize location diversity, will motivate wide-scale distribution even if such distribution comes at an economic cost. Therefore, based on the data we collected, we conclude that PlanetLab applications *must* time-share nodes, because it is unlikely that a platform like PlanetLab would

ever grow to a size that could accommodate even a few applications each "owning" hundreds of nodes on an ongoing basis, not to mention allowing those large-scale applications to coexist with multiple smaller-scale experiments. And it is this best-effort time-sharing of node resources that leads to variable per-node resource availability over time on PlanetLab.

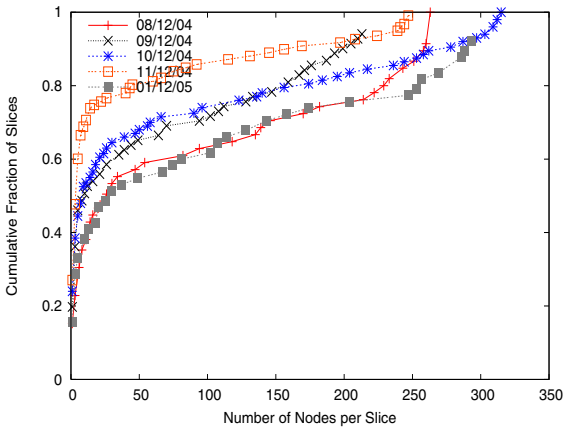


**Figure 1: Cumulative distribution of nodes per slice averaged over the first day of each month.**

At the same time, more than 70% of slices run on less than half of the nodes, suggesting significant flexibility in mapping slices to nodes. Of course, even applications that wish to run on all platform nodes may benefit from intelligent resource selection, by mapping their most resource-intensive slivers to the least utilized nodes and vice-versa.

Having argued for the necessity of time-sharing nodes, the remainder of this section quantifies resource variability across nodes and investigates the potential for real applications to exploit that heterogeneity by making informed node selection decisions at deployment time. We observe significant variability of available resources across nodes, and we find that in simulation, a simple load-sensitive sliver placement algorithm outperforms a random placement algorithm. These observations suggest that some applications are likely to benefit from informed placement at deployment time.

## 3.1 Resource heterogeneity across nodes

Table 1 shows that static per-node PlanetLab node characteristics are fairly homogeneous across nodes. However, dynamic resource demands are heavy and vary over both space and time. To quantify such variation, we consider the 5-minute load average on all PlanetLab nodes at three instants in our 6-month trace. These instants correspond to an instant of low overall platform utilization, an instant of typical platform utilization, and an instant of high platform utilization. Figure 2 shows a CDF of the

| attribute | mean | std. dev. | median | 10th %ile | 90th %ile |
|---|---|---|---|---|---|
| # of CPUs | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 |
| cpu speed (MHz) | 2185 | 642.9 | 2394 | 1263 | 3066 |
| total disk (GB) | 98 | 48 | 78 | 69 | 156 |
| total mem (MB) | 1184 | 364 | 1036 | 1028 | 2076 |
| total swap (GB) | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 |

**Table 1: Static node measurements on Feb 18, 2005. Similar results were found during all other time periods as well. PlanetLab nodes are also geographically diverse, located in over 30 countries.**

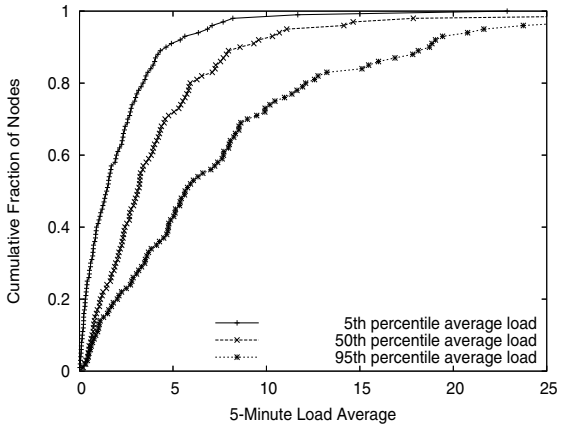


**Figure 2: CDF of node loads at three representative moments in the trace: when load averaged across all nodes was "typical" (median), "low" (5th percentile), and "high" (95th percentile).**

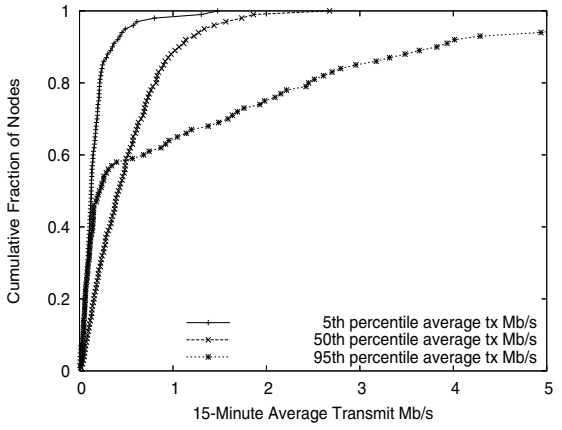


**Figure 3: CDF of node 15-minute transmit bandwidths at three representative moments in the trace: when node transmit bandwidth averaged across all nodes was "typical" (median), "low" (5th percentile), and "high" (95th percentile). We found similar results for 15-minute receive bandwidth.**

5-minute load average across all nodes at three separate moments in time: the time when the load averaged across all nodes was the 5th percentile of such averages over the entire trace (low platform utilization), the time when the load averaged across all nodes was the the median of such averages over the entire trace (typical platform utilization), and the time when the load averaged across all nodes was the 95th percentile of such averages over the entire trace (high platform utilization). Analogously, Figure 3 shows a CDF of the 15-minute average transmit bandwidth across all nodes at three moments: low, typical, and high utilization of platform-wide transmit bandwidth. Results for receive bandwidth were similar.

We see that load varies substantially across nodes independent of overall system utilization, while network bandwidth utilization varies substantially across nodes primarily when the platform as a whole is using significantly higher-than-average aggregate bandwidth. This suggests that load is always a key metric when making application placement decisions, whereas available per-node network bandwidth is most important during periods of peak platform bandwidth utilization. Note that periods of low platform bandwidth utilization may be due to low aggregate application demand or due to inability of the network to satisfy demand.

Not only per-node attributes, but also inter-node characteristics, vary substantially across nodes. Figure 4 shows significant diversity in latency and available bandwidth among pairs of PlanetLab nodes.

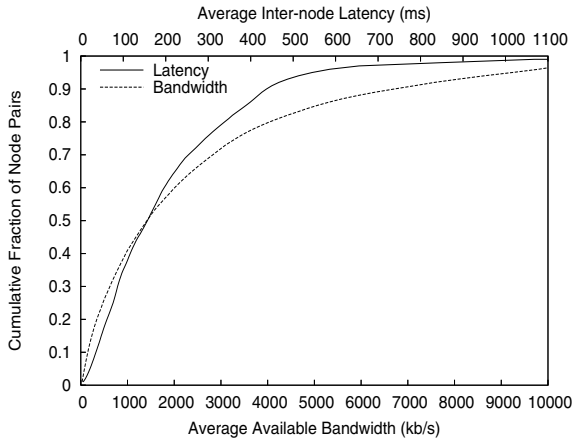


**Figure 4: CDF of node-pair latencies and available bandwidths over the entire trace.**

Furthermore, we find that PlanetLab nodes exhibit a wide range of MTTFs and MTTRs. Figure 5 shows MTTF and MTTR based on "uptime" measurements recorded by nodes. We declare that a node has failed when its uptime decreases between two measurement intervals. We declare the node to have failed at the time we received the last measurement report with a monotonically increasing uptime, and to have recovered at $t_{report} - \Delta t_{up}$ where $t_{report}$ is the time we receive the measurement report indicating an uptime less than the previous measurement report, and $\Delta t_{up}$ is the uptime measurement in that report. Computing MTTF and MTTR based on "all-pairs pings" data, with a node declared dead if no other node can reach it, yields similar results. Compared to the uptime-based approach, MTTF computed using pings is slightly lower, reflecting situations where a node becomes disconnected from the network but has not crashed, while MTTR computed using pings is slightly higher, reflecting situations in which some time passes between a node's rebooting and the restarting of network services.

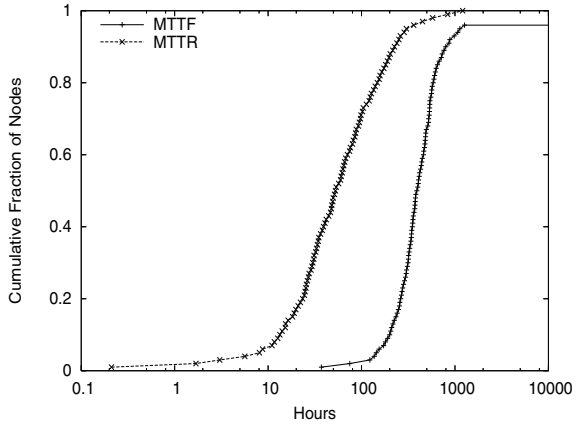


**Figure 5: CDF of MTTF and MTTR based on "uptime." These curves show median availability of 88% and 95th percentile availability of 99.7%.**

These observations of substantial heterogeneity across node and node-pair attributes suggest that there is much available resource diversity for a service placement mechanism to exploit.

## 3.2 Nodes meeting application requirements

Having observed significant heterogeneity across nodes, we now examine how that heterogeneity combines with the resource requirements of real applications to dictate the potential usefulness of informed application placement. In order to do this, we first must develop models of these applications' resource demands. In particular, we study resource usage by three long-running PlanetLab services.

1. **CoDeeN** [26] is a prototype Content Distribution Network. It has been operating since June 2003 and currently receives about 10 million hits per day from about 30,000 users. Over the course of the trace, CoDeeN slivers ran on 384 unique nodes.

2. **Coral** [10] is another Content Distribution Network. It has been operating since August 2004 and cur-

rently receives 10-20 million hits per day from several hundred thousand users. Over the course of the trace, Coral slivers ran on 337 unique nodes.

3. **OpenDHT** [20] is a publicly accessible distributed hash table. OpenDHT has been operating since mid-2004. Over the course of the trace, OpenDHT slivers ran on 406 unique nodes.

We chose these services because they are continuously-running, utilize a large number of PlanetLab nodes, and serve individuals outside the computer science research community. They are therefore representative of "production" services that aim to maximize user-perceived performance while coexisting with other applications in a shared distributed infrastructure. They are also, to our knowledge, the longest-running non-trivial PlanetLab services. Note that during the time period of our measurement trace, these applications underwent code changes that affect their resource consumption. We do not distinguish changes in resource consumption due to code upgrades from changes in resource consumption due to workload changes or other factors, as the reason for a change in application resource demand is not important in addressing the questions we are investigating, only the effect of the change on application resource demand.

We study these applications' resource utilization over time and across nodes to develop a loose model of the applications' resource needs. We can then combine that model with node-level resource data to evaluate the quality of different mappings of slivers (application instances) to nodes. Such an evaluation is essential to judging the potential benefit of service placement and migration.

Figures 6, 7, and 8 show CPU demand and outgoing network bandwidth for these applications over time, using a log-scale Y-axis. Due to space constraints we show only a subset of the six possible graphs, but all graphs displayed similar characteristics. Each graph shows three curves, corresponding to the 5th percentile, median, and 95th percentile resource demand across all slivers in the slice at each timestep. (For example, the sliver whose resource demand is the 95th percentile at time $t$ may or may not be the same sliver whose resource demand is the 95th percentile at time $t + n$.)

We see that a service's resource demands vary significantly across slivers at any given point in time. For example, all three graphs show resource demand varying by at least an order of magnitude from the median sliver to the 95th percentile sliver. These results suggest that not only available host resources, but also variable per-sliver resource demand, must be considered when making placement decisions.

One interpretation of widely varying per-sliver demands is that applications are performing internal load balancing, i.e., assigning more work to slivers running on nodes

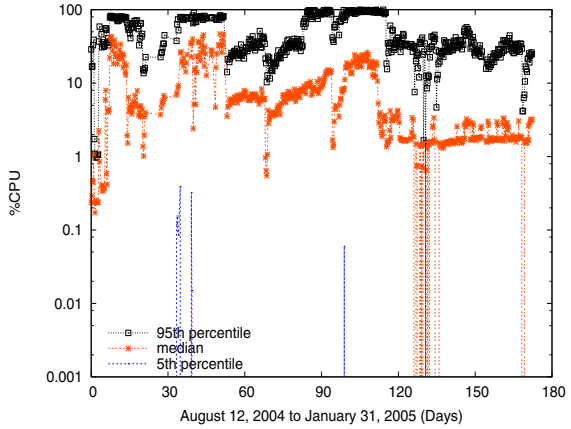


**Figure 6: CPU resource demand for OpenDHT, using a log-scale Y-axis.**

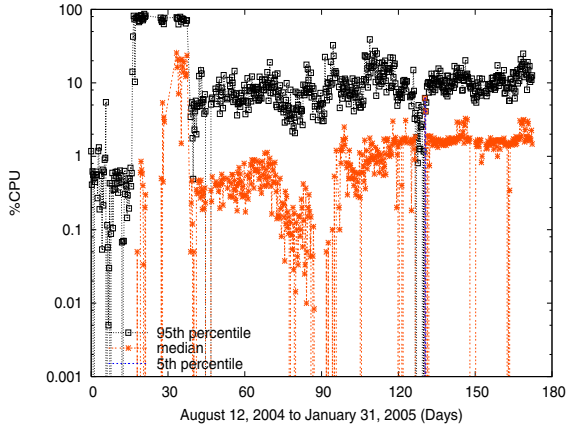


**Figure 7: CPU resource demand for Coral, using a log-scale Y-axis.**

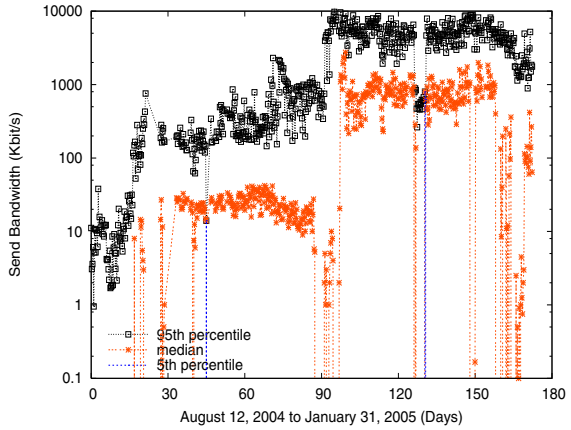


**Figure 8: Transmit bandwidth demand for Coral, using a log-scale Y-axis.**

with ample free resources. However, these three applications principally use a hash of a request's contents to map requests to nodes, suggesting that variability in per-node demands is primarily attributable to the application's external workload and overall structure, not internal load balancing. For instance, OpenDHT performs no internal load balancing; Coral's Distributed Sloppy Hash Table balances requests to the same key ID but this balancing does not take into account available node resources; and CoDeeN explicitly takes node load and reliability into account when choosing a reverse proxy from a set of candidate proxies.

Given this understanding of the resource demands of three large-scale applications, we can now consider the potential benefits of informed service placement. To answer this question, we simulate deploying a *new* instance of one of our three modeled applications across PlanetLab. We assume that the new instance's slivers will have resource demands identical to those of the original instance. We further assume that that the new instance will use the same set of nodes as the original, to allow for the possibility that the application deployer intentionally avoided certain nodes for policy reasons or because of known availability problems. Within these constraints, we allow for any possible one-to-one mapping of slivers to nodes. Thus, our goal is to determine how well a particular service placement algorithm will meet the requirements of an application, knowing both the application's resource demands and available resources of the target infrastructure.

We simulate two allocation algorithms. The *random* algorithm maps slivers to nodes randomly. The *load-sensitive* algorithm deploys heavily CPU-consuming slivers onto lightly loaded nodes and vice-versa. In both cases, each sliver's resource consumption is taken from the sliver resource consumption measurements at the corresponding timestep in the trace, and each node's amount of free resources is calculated by applying the formulas discussed in Section 2 to the node's load and bandwidth usage indicated at that timestep in the trace. We then calculate, for each timestep, the fraction of slivers whose assigned nodes have "sufficient" free CPU and network resources for the sliver, as defined in Section 2. If our *load-sensitive* informed service placement policy is useful, then it will increase, relative to random placement, the fraction of slivers whose resource needs are met by the nodes onto which they are deployed. Of course, if a node does not meet a sliver's resource requirements, that sliver will still function from a practical standpoint, but its performance will be impaired.

Figures 9 and 10 show the fraction of nodes that meet application CPU and network resource needs, treating each timestep in the trace as a separate deployment. As Figures 2 and 3 imply, CPU was a more significant bottleneck than node access link bandwidth. We see that the load-sensitive placement scheme outperforms the random placement scheme, increasing the number of slivers running on nodes that meet their resource requirements by as much as 95% in the case of OpenDHT and as much as 55% in the case of Coral (and CoDeeN, the graph of which is omitted due to space constraints). This data argues that there is potentially significant performance improvement to be gained by using informed service placement based on matching sliver resource demand to nodes with sufficient available resources, as compared to a random assignment. A comprehensive investigation of what application characteristics make informed node selection more beneficial or less beneficial for one application compared to another is left to future work.

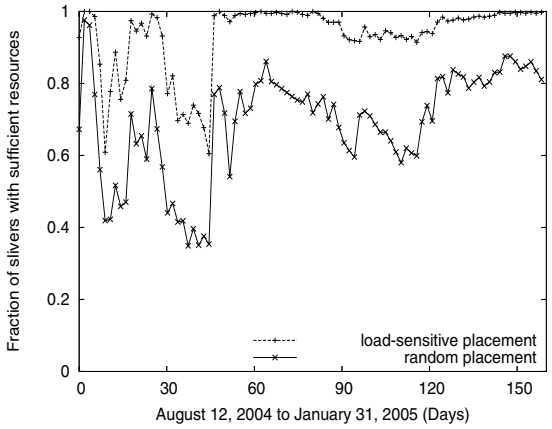


**Figure 9: Fraction of OpenDHT slivers whose resource requirements are met by the node onto which they are deployed, vs. deployment time.**

## 4. SLIVER MIGRATION

We have seen that applications can potentially benefit from intelligently mapping their slivers to nodes based on sliver resource demand and available node resources. Our next question is whether migrating slivers could improve overall application performance—that is, whether, and how often, to periodically recompute the mapping. While process migration has historically proven difficult, many distributed applications are designed to gracefully handle node failure and recovery; for such applications, migration requires simply killing an application instance on one node and restarting it on another node. Furthermore, emerging virtual machine technology may enable low-overhead migration of a sliver without resorting to exit/restart. Assuming the ability to migrate slivers, we consider the potential benefits of doing so in this section. Of course, migration is feasible only for services that do not need to "pin" particular slivers to particular nodes. For example, sliver location is not "pinned" in services

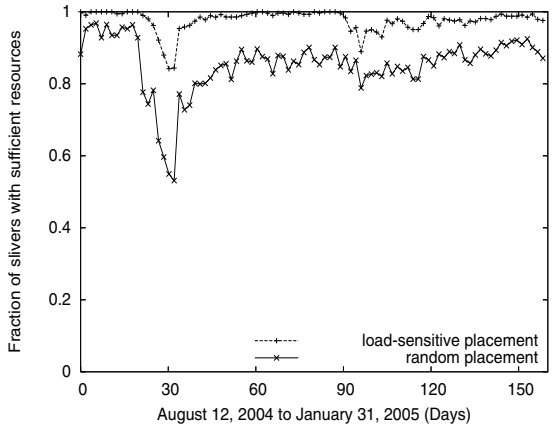


**Figure 10: Fraction of Coral slivers whose resource requirements are met by the node onto which they are deployed, vs. deployment time. We note that Coral's relatively low per-sliver CPU resource demands result in a larger fraction of its slivers' resource demands being met relative to OpenDHT.**

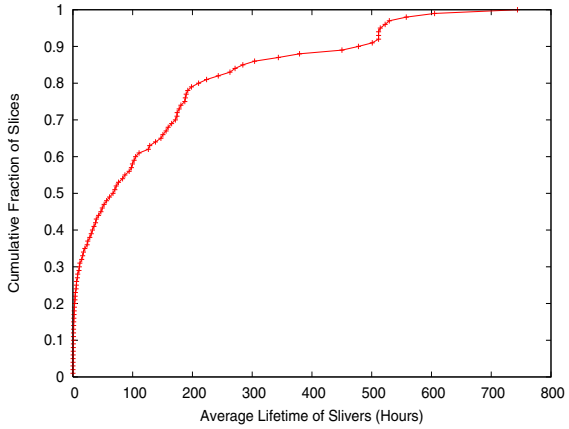


**Figure 11: CDF of fraction of slices vs. average sliver lifetime for that slice. A sliver may die due to software failure or node crash; when the sliver comes back up, we count it as a new sliver.**

that map data to nodes pseudo-randomly by hashing the contents of requests, as is the case (modulo minor implementation details) for the three typical PlanetLab applications we have studied. We comment on the applicability of these results to additional application classes in Section 7.

Before considering the potential benefits of migration, we must first determine the typical lifetime of individual slivers. If most slivers are short-lived, then a complex migration infrastructure is unnecessary since per-node resource availability and per-sliver resource demand are unlikely to change significantly over very short time scales. In that case making sliver-to-node mapping decisions only when slivers are instantiated, i.e., when the application is initially deployed and when an existing sliver dies and must be re-started, should suffice. Figure 11 shows the average sliver lifetime for each slice in our trace. We see that slivers are generally long-lived: 75% of slices have average sliver lifetimes of at least 6 hours, 50% of slices have average sliver lifetimes of at least two days, and 25% of slices have average sliver lifetimes of at least one week. As before, we say that a sliver is "alive" on a node if it appears in the process table for that node.

To investigate the potential usefulness of migration, we next examine the rate of change of available node resources. If per-node resource availability varies rapidly relative to our measurements of sliver lifetimes, we can hypothesize that sliver migration may be beneficial.

## 4.1 Node resource variability over time

To assess the variability of per-node available resources over time, we ask what fraction of nodes that meet a particular resource requirement at time $T$ continue to meet that requirements for all time intervals between $T$ and $T+x$, for various values of $x$ and all times $T$ in our trace. If the fraction is large, then most slivers initially deployed to nodes meeting the requirement at time $T$ will find themselves continuously executing on nodes that meet the requirement until time $T+x$. Conversely, if the fraction is small, then most slivers will find the resource requirement violated at some time before $T+x$, suggesting that they may benefit from migration at a time granularity on the order of $x$.

Figures 12 and 13 show the fraction of nodes that meet a particular resource requirement at time $T$ that continue to meet the requirement for all time intervals between $T$ and $T+x$, for various values of $x$, averaged over all starting times $T$ in our trace. The fraction of nodes that continually meet initial requirements declines rapidly with increasing intervals $x$, and the rate of decline increases with the stringency of the requirement. Most importantly, we see that the fraction of nodes continually meeting typical resource requirements remains relatively high (80% or greater) up to about 30 minutes post-deployment for load and up to about 60 minutes post-deployment for network traffic. This result suggests that if sliver resource requirements remain relatively constant over time, then it is unnecessary to migrate more often than every 30 to 60 minutes.

## 4.2 Sliver suitability to nodes over time

The suitability of a particular node to host a particular sliver depends not only on the resources available on that node, but also on the resource demands of that sliver over time. We therefore perform an analysis similar to that in the previous section, but accounting for

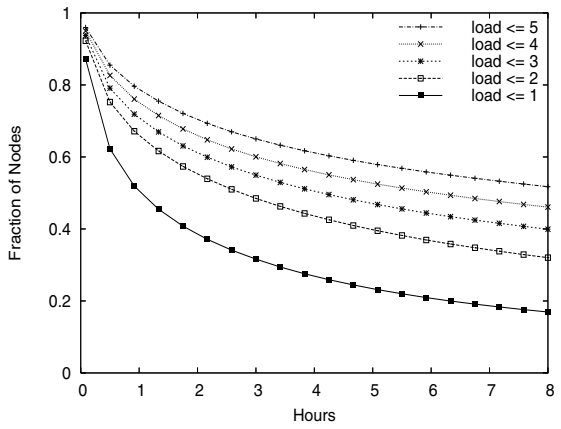


**Figure 12: Fraction of nodes continuously meeting various load constraints for various durations after initially meeting the constraint. The fraction is 100% at $x = 0$ because we consider only nodes that initially meet the constraint.**

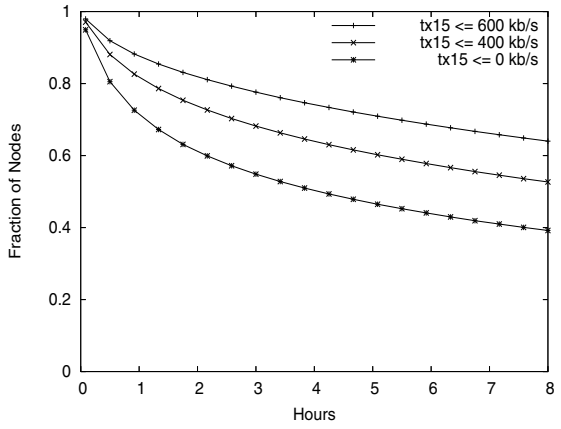


**Figure 13: Fraction of nodes continuously meeting various constraints on network transmit bandwidth from competing applications for various durations after initially meeting the constraint. The fraction is 100% at $x = 0$ because we consider only nodes that initially meet the constraint. Similar results were found for receive bandwidth.**

both available node resources and application resource demand. Here we are interested not in the stability of available resources on individual nodes, but rather in the stability of the fraction of slivers whose resource requirements are met after deployment. It is the rate of decline of this fraction that dictates an appropriate migration interval for the application—very rapid decline will require prohibitively frequent migration, while very slow decline means migration will add little to a simple policy of intelligent initial sliver placement and re-deployment upon failure.

Thus, we ask what fraction of nodes onto which slivers are deployed at time $T$ meet the requirements of their sliver at time $T + x$, for various values of $x$. For each $T + x$ value, we average this measure over every possible deployment time $T$ in our trace. A large fraction means that most slivers will be running on satisfactory hosts at the corresponding time. As in Section 3.2, we say a node meets a sliver's requirements if the node has enough free CPU and network bandwidth resources to support a new sliver assigned from the set of sliver resource demands found at that timestep in the trace, according to the load-sensitive or random placement policy.

Figures 14 and 15 show the fraction of slivers whose resource requirements were met at the time indicated on the X-axis, under both the random and load-sensitive schemes for initially mapping slivers to nodes at time $X = 0$. Note that the random placement line is simply a horizontal line at the value corresponding to average across all time intervals from Figure 9 in the case of OpenDHT and Figure 10 in the case of Coral.

We make two primary observations from these graphs. First, the quality of the initially load-sensitive assignment degrades over time as node resources and sliver demands become increasingly mismatched. This argues for periodic migration to re-match sliver needs and available host resources. Second, the benefit of load-sensitive placement over random placement—the distance between the load–sensitive and random lines—erodes over time for the same reason, but persists. This persistence suggests that informed initial placement can be useful even in the absence of migration.

Choosing a desirable migration period requires balancing the cost of migrating a particular application's slivers against the rate at which the application mapping's quality declines. For example, in OpenDHT, migration is essentially "free" since data is stored redundantly—an OpenDHT instance can be killed on one node and re-instantiated on another node (and told to "own" the same DHT key range as before) without causing the service to lose any data. Coral and CoDeeN can also be migrated at low cost, as they are "soft state" services, caching web sites hosted externally to their service. An initially load-sensitive sliver mapping for OpenDHT has declined to close to its asymptotic value within 30 minutes, arguing for migrating poorly-matched slivers at that timescale or less. If migration takes place every 30 minutes, then the quality of the match will, on average, traverse the curve from $t = 0$ to $t = 30$ every 30 minutes, returning to $t = 0$ after each migration. Coral and CoDeeN placement quality declines somewhat more quickly than OpenDHT, but migrating poorly matched slivers of these services on the order of every 30 minutes is unlikely to cause harm and will allow the system to maintain a somewhat better mapping than would be achieved with a less aggressive

migration interval.

A comprehensive investigation of what application characteristics make migration more beneficial or less beneficial for one application compared to another is left to future work, as is emulation-based verification of our results (i.e., implementing informed resource selection and migration in real PlanetLab applications, and measuring user-perceived performance with and without those techniques under repeatable system conditions). Our focus in this paper is a simulation-based analysis of whether designers of future resource selection systems should consider including informed placement and migration capabilities, by showing that those techniques are potentially beneficial for several important applications on a popular existing platform.

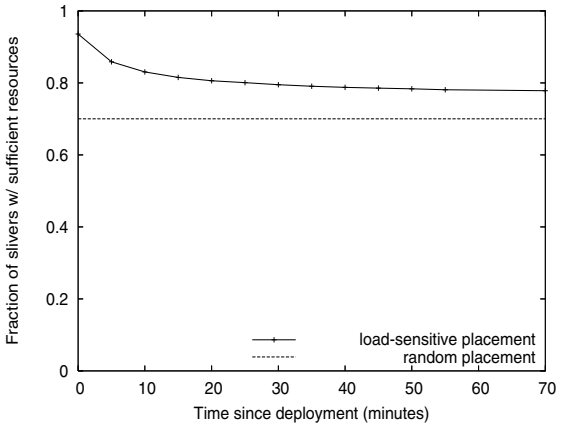


**Figure 14: Fraction of OpenDHT slivers hosted on nodes that meet the sliver's requirements at the time that is indicated on the X-axis.**

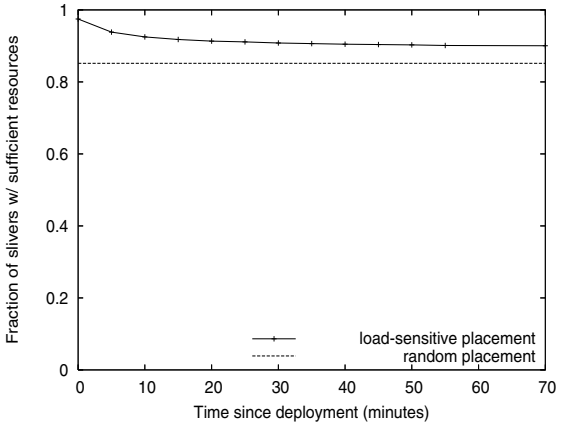


**Figure 15: Fraction of Coral slivers hosted on nodes that meet the sliver's requirements at the time that is indicated on the X-axis. CoDeeN showed similar results.**

# 5. DESIGN OPTIMIZATIONS

Our preceding experiments have assumed that resource availability data is collected from every node every 5 minutes, the minimum time granularity of our trace. In a large-scale system, it may be undesirable to collect data about every resource attribute from all nodes that frequently. Thus, we investigate two optimizations that a service placement and migration service might use to reduce measurement overhead. First, the system might simply collect node measurement data less frequently, accepting the tradeoff of reduced accuracy. Second, it might use statistical techniques to predict resource values for one resource based on measurements it has collected of other resource values on the same node, resource values on other nodes, or historical resource values.

## 5.1 Reducing measurement frequency

In this section, we examine the impact of reducing measurement frequency, first by quantifying the relative measurement error resulting from relaxed data measurement intervals, and then by studying the impact that stale data has on migration decisions for our modeled applications.

### *5.1.1 Impact on measurement accuracy*

Figures 16, 17, and 18 show the accuracy impact of relaxed measurement intervals for load, network transmit bandwidth, and inter-node latency. For each of several update intervals longer than 5 minutes, we show the fraction of nodes (or, in the case of latency, node pairs) that incur various average absolute value errors over the course of the trace, compared to updates every 5 minutes (15 minutes for latency).

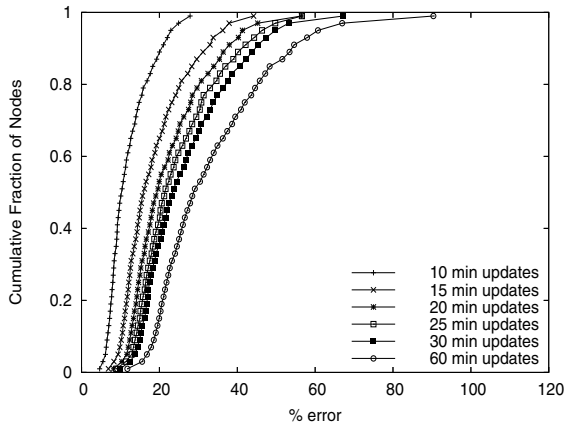


**Figure 16: Mean error in 5-minute load average compared to 5-minute updates.**

We observe several trends from these graphs. First, latency is more stable than load or network transmit bandwidth. For example, if a maximum error of 20% is tolerable, then moving from 15-minute measurements to hourly

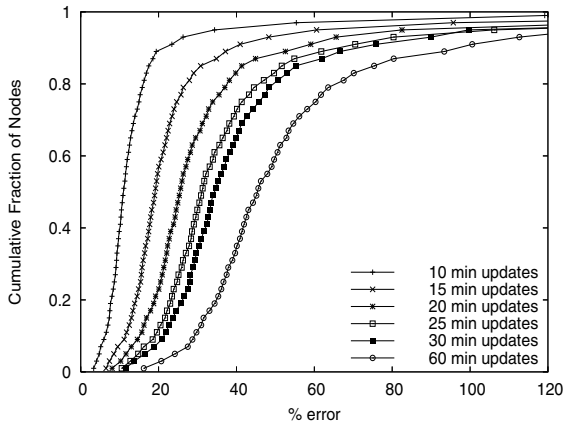


**Figure 17: Mean error in 15-minute average network transmit bandwidth compared to 5-minute updates. Similar results were found for receive bandwidth.**

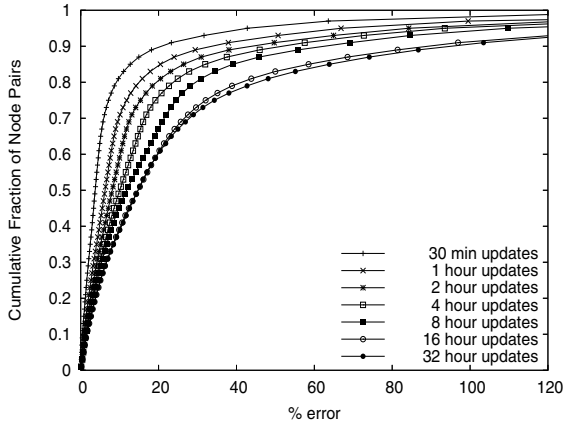


**Figure 18: Mean error in inter-node latency compared to 15-minute updates.**

measurements for latency will push about 12% of node pairs out of the 20% accuracy range, but moving from 15-minute measurements to hourly measurements for load and network bandwidth usage will push about 50% and 55% of nodes, respectively, out of the 20% accuracy range. Second, network latency, and to a larger extent network bandwidth usage, show longer tails than does load. For example, with a 30-minute update interval, only 3% of nodes show more than 50% error in load, but over 20% of nodes show more than 50% error in network bandwidth usage. This suggests that bursty network behavior is more common than bursty CPU load.

From these observations, we conclude that for most nodes, load, network traffic, and latency data can be collected at relaxed update intervals, e.g., every 30 minutes, without incurring significant error. Of course, the exact amount of tolerable error depends on how the measurement data is being used. Further, we see that a small number of nodes show significant burstiness with respect to network behavior, suggesting that a service placement infrastructure could maximize accuracy while minimizing overhead by separating nodes based on variability of those attributes, and using a relaxed update rate for most nodes but a more aggressive one for nodes with high variance.

### *5.1.2 Impact on placement and migration decisions*

Next we investigate the impact that error resulting from relaxed measurement intervals would have on the decisions made by a service placement infrastructure. Figures 14 and 15 already contain the answer to this question, as there is an analogy between acceptable migration interval and acceptable data staleness. For example, consider collecting measurements every 30 minutes. If a migration decision is made at the same time as the data is collected, then on average the quality of the sliver-to-node mapping will be the value of the curve at $t = 0$. If a migration decision is made 5 minutes later, but using the same measurements, then the quality of the sliver-to-node matching will be the value of the curve at $t = 5$. And so on, up to $t = 29$. This analogy leads us to a similar conclusion regarding stale data as we made regarding migration interval, namely that data staleness up to 30 minutes is acceptable for making migration decisions for this workload.

## 5.2 Predicting node resources

In this section, we investigate whether we can predict the availability of a resource on a host using values for other resources on the same host, the same resource on other hosts, or earlier measurements of the same resource on the same host. If such correlations exist, a placement and migration service can reduce measurement overhead by collecting only a subset of the measurements that are needed, and inferring the rest.

### *5.2.1 Correlation among attributes*

We first investigate the correlation among attributes on the same node. A high correlation would allow us to use the value of one attribute on the node to predict the values of other attributes on the node. Table 2 shows the correlation coefficient (r) among attributes on the same node, based on data from all nodes and all timesteps in our trace. Somewhat surprisingly, we see no strong correlations—we might expect to see a correlation between load and network bandwidth, free memory, or swap space. Instead, because each PlanetLab node is heavily multiprogrammed, as suggested by Figure 1, the overall resource utilization is an average (aggregate) across many applications. A spike in resource consumption by one application might occur at the same time as a dip on

resource consumption by another application, leaving the net change "in the noise." We found a similar negative result when examining the correlation of a single attribute across nodes at the same site (e.g., between load on pairs of nodes at the same site). While we initially hypothesized that there may be some correlation in the level of available resources within a site, for instance because of user preference for some geographic or network locality, the weakness of these same-site correlations implies that we cannot use measurements of a resource on one node at a site to predict values of that resource on other nodes at the site.

| r | load | mem | swapfree | bytes_in | bytes_out |
|---|---|---|---|---|---|
| **load** | | | | | |
| **mem** | -0.04 | | | | |
| **swapfree** | -0.26 | 0.18 | | | |
| **bytes_in** | 0.17 | -0.062 | -0.20 | | |
| **bytes_out** | 0.08 | -0.077 | 0.01 | 0.44 | |

**Table 2: Correlation between pairs of attributes on the same node: 15-minute load average, free memory, free swap space, 15-minute network receive bandwidth, and 15-minute network transmit bandwidth.**

One pair of potentially correlated attributes that merits special attention is inter-node latency and bandwidth. In general, for a given loss rate, one expects bandwidth to vary roughly with $1/latency$ [16]. If we empirically find a strong correlation between latency and bandwidth, we might use latency as a surrogate for bandwidth, saving substantial measurement overhead.

To investigate the potential correlation, we annotated each pairwise available bandwidth measurement collected by Iperf with the most recently measured latency between that pair of nodes. We graph these $(latency, bandwidth)$ tuples in Figure 19. Fitting a power law regression line, we find a correlation coefficient of -0.59, suggesting a moderate inverse power correlation. One reason why the correlation is not stronger is the presence of nodes with limited bandwidth (relative to the bulk of other nodes), such as DSL nodes and nodes configured to limit outgoing bandwidth to 1.5 Mb/s or lower. These capacity limits artificially lower available bandwidth below what would be predicted based on the latency-bandwidth relationship from the dataset as a whole. Measurements taken by nodes in this category correspond to the dense rectangular region at the bottom of Figure 19 below a horizontal line at 1.5 Mb/s, where decreased latency does not correlate to increased bandwidth.

When such nodes are removed from the regression equation computation, the correlation coefficient improves to a strong -0.74. Viewed another way, using a regression equation derived from all nodes to predict available bandwidth using measured latency leads to an average 233% error across all nodes. But if known bandwidth-limited nodes are excluded when computing the regression equation, predicting available bandwidth using measured latency leads to only an average 36% error across the non-bandwidth-limited nodes. Additionally, certain node pairs show even stronger latency-bandwidth correlation. For example, 48% of node pairs have bandwidths within 25% of the value predicted from their latency. We conclude that a power-law regression equation computed from those nodes with "unlimited" bandwidth (not DSL or administratively limited) allows us to accurately predict available bandwidth using measured latency for the majority of those non-bandwidth-limited nodes. This in turn allows a resource discovery system to reduce measurement overhead by measuring bandwidth among those nodes infrequently (only to periodically recompute the regression equation), and to use measured latency to estimate bandwidth the rest of the time. Of course, if the number of bandwidth-capped nodes in PlanetLab increases, then more nodes would have to be excluded and this correlation would become of less value.

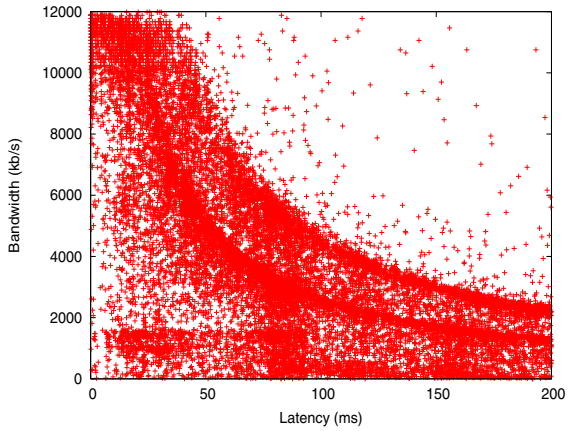


**Figure 19: Correlation between latency and available bandwidth. Each point represents one end-to-end bandwidth measurement and the latency measurement between the same pair of nodes taken at the closest time to that of the bandwidth measurement.**

### 5.2.2 *Predictability over time*

Finally we investigate the predictability of host resources over time. We focus on predicting host 5-minute load average and 15-minute bandwidth average, over periods of 5 minutes and 1 hour.

The most well-known step-ahead predictors in the context of wide-area platforms are those implemented in the Network Weather Service (NWS) [29]. Although originally designed to predict network characteristics, they have also been used to predict host CPU load [31]. We consider the following NWS predictors: last value, exponentially-weighted moving average (EWMA), me-

dian, adaptive median, sliding window average, adaptive average, and running average. For each prediction strategy and host, we compute the average absolute value prediction error for that host across all time intervals, and the standard deviation of the prediction errors for that host. Table 3 shows the average bandwidth prediction error and average load prediction error for the median host using the three NWS techniques that performed best for our dataset (last, EWMA, and median). We also show results for a "dynamic tendency" predictor, which predicts that a series of measurements that has been increasing in the recent past will continue to increase, and a series that has been decreasing in the recent past will continue to decrease [31]. The 5-minute and one-hour predictors operate identically except that the input to the one-hour predictors is the average value over each one-hour period, while the input to the 5-minute predictor is each individual measurement in our trace.

We find that the "last value" predictor performs well over time periods of an hour, confirming and extending our findings from Figures 12 and 13 that load and network bandwidth usage remain relatively stable over periods of an hour. However, as we can see from Figures 2 and 3, the system undergoes dramatic and unpredictable resource demand variations over longer time scales.

The "dynamic tendency" and "last value" predictors perform the best of all predictors we considered, for the following reason. All other predictors (EWMA, median, adaptive median, sliding window average, adaptive average, and running average) predict that the next value will return to the mean or median of some multi-element window of past values. In contrast, the dynamic tendency predictor predicts that the next value will continue along the trend established by the multi-element window of past values. The "last value" predictor falls between these two policies: it keeps just one element of state, predicting simply that the next value will be the same as the last value. PlanetLab load and network utilization values tend to show a mild tendency-based pattern—if the load (or network utilization) on a node has been increasing in the recent past, it will tend to continue increasing, and vice-versa. As a result, the dynamic tendency and last value predictors perform the best. Our results resemble those in [31], which showed errors in the 10-20% range for both last-value and dynamic tendency predictors, in a study of loads from more traditional servers.

Analogous to using a machine's last load value as a prediction of its next load value, we might use a node's historical MTTF (MTTR) to predict its future MTTF (MTTR). To evaluate the effectiveness of this technique, we split the trace of node failures and recoveries that we used in Section 3.1 into two halves. For each node, we calculate its MTTF and MTTR during the first half of the trace. We predict that its MTTF (MTTR) during the second half will be the same as its MTTF (MTTR) during the first half, and calculate the percentage error that this prediction yields. Figure 20 shows a CDF of the fraction of nodes for which this predictor yields various prediction errors. We find substantial prediction error (greater than 100%) for only about 20% of nodes, suggesting that historical node MTTF and MTTR are reasonable criteria for ranking the quality of nodes when considering where to deploy an application. On the other hand, this prediction technique does not yield extremely accurate predictions—for MTTF, error for the median node is 45%, and for MTTR, error for the median node is 87%.

| attribute | prediction technique | 5-minute prediction error | 1-hour prediction error |
|---|---|---|---|
| load | dynamic tend. | 17.8% | 23.5% |
| load | last | 17.8% | 23.5% |
| load | EWMA | 22.4% | 30.7% |
| load | median | 24.3% | 35.5% |
| net bw | dynamic tend. | 29.5% | 34.6% |
| net bw | last | 29.5% | 34.6% |
| net bw | EWMA | 42.7% | 48.7% |
| net bw | median | 36.8% | 46.2% |

**Table 3: 5-minute and 1-hour median per-host average prediction error of best-performing NWS predictors and the dynamic tendency predictor.**

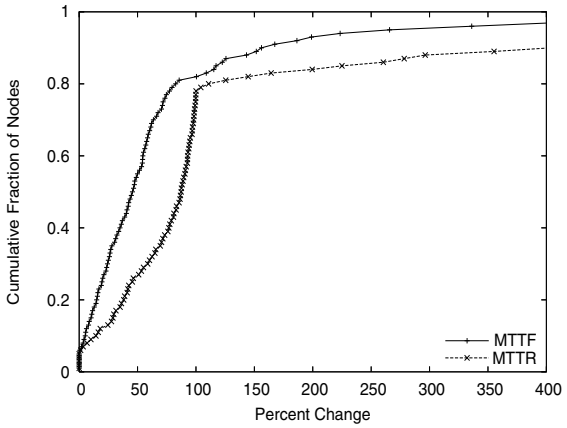


**Figure 20: Prediction error for MTTF and MTTR.**

Finally, we examine the periodicity of resource availability. Periodicity on the time scale of human schedules is a common form of medium-term temporal predictability. It is often found in utilization data from servers hosting applications with human-driven workloads. For example, a web site might see its load dip when it is nighttime in the time zones where the majority of its users reside, or on weekends. We therefore examined our PlanetLab traces for periodicity of per-node load and network bandwidth usage over the course of a day and week. We found no such periodicity for either attribute. In fact, on average, the load or network bandwidth on a node at time

$t$ was less closely correlated to its value at time $t + 24$ hours than it was to its value at a random time between $t$ and $t + 24$ hours. Likewise, on average, the load or network bandwidth on a node at time $t$ was less closely correlated to its value at time $t + 1$ week than it was to its value at a random time between $t$ and $t + 1$ week.

The lack of daily and weekly periodicity on PlanetLab can be explained by the wide geographic distribution of application deployers and the users of the deployed services such as those studied earlier in this paper. Further, load on PlanetLab tends to increase substantially around conference deadlines, which happen on yearly timescales (beyond the granularity of our trace) rather than daily or weekly ones. In sum, we find that resources values are more strongly correlated over short time periods than over medium or long-term ones.

## 6. RELATED WORK

The measurement aspects of this paper add to a growing literature on measurements of resource utilization in Internet-scale systems. Most of this work has focused on network-level measurements, a small subset of which we mention here. Balakrishnan examines throughput stability to many hosts from the vantage point of the 1996 Olympic Games web server [2], while Zhang collects data from 31 pairs of hosts [34]. Chen describes how to monitor a subset of paths to estimate loss rate and latency on all other paths in a network [6] . Wolski [29] and Vazhkudai [25] focus on predicting wide-area network resources.

Growing interest in shared computational Grids has led to several recent studies of node-level resource utilization in such multi-user systems. Foster describes resource utilization on Grid3 [8], while Yang describes techniques to predict available host resources to improve resource scheduling [31, 32]. Harchol-Balter investigates process migration for dynamic load-balancing in networks of Unix workstations [12], and cluster load balancing is an area of study with a rich literature. Compared to these earlier studies, our paper represents the first study of resource utilization and service placement issues for a federated platform as heavily shared and utilized as PlanetLab.

Several recent papers have used measurement data from PlanetLab. Yalagandula investigates correlated node failure; correlations between MTTF, MTTR, and availability; and predictability of TTF, TTR, MTTF, and MTTR [30]. Rhea measures substantial variability over time and across nodes in the amount of time to complete CPU, disk, and network microbenchmarks; these findings corroborate our observations in Section 3.1 [19]. Rhea advocates application-internal mechanisms, as opposed to intelligent application placement, to counter node heterogeneity. Lastly, Spring uses measurements of CPU and node availability to dispel various "myths" about PlanetLab [23].

Resource discovery tools are a prerequisite for automated service placement and migration. CoMon [18], CoTop [17], and Ganglia [14] collect node-level resource utilization data on a centralized server, while MDS [33], SWORD [15], and XenoSearch [22] provide facilities to query such data to make placement and migration decisions.

Shared wide-area platforms themselves are growing in number and variety. PlanetLab focuses on network services [3], Grid3 focuses on large-scale scientific computation [8], FutureGrid aims to support both "eScience" and network service applications [7], and BOINC allows home computer users to multiplex their machines' spare resources among multiple public-resource computing projects [4]. Ripeanu compares resource management strategies on PlanetLab to those used in the Globus Grid toolkit [21].

## 7. CONCLUSIONS AND FUTURE WORK

Resource competition is a fact of life when time-shared distributed platforms attract a substantial number of users. In this paper, we argued that careful application placement and migration are promising techniques to help mitigate the impact of resource variability resulting from this competition. We also studied techniques to reduce measurement overhead for a placement and migration system, including using stale or predicted data.

Resource selection and application migration techniques complement the application-specific techniques that some distributed services employ internally to balance load or to select latency-minimizing network paths. Those techniques optimize application performance given the set of nodes already supporting the application, and generally only consider the application's own workload and structure as opposed to resource constraints due to competing applications. In contrast, this paper focused on *where to deploy—and possibly re-deploy—application instances* based on information about application resource demand and available node and network resources. Once an application's instances have been mapped to physical nodes, application-internal mechanisms can then be used on finer timescales to optimize performance. In general, application-internal load balancing, external service placement, or a combination of the two can be used to match application instance to available nodes based on resource demand and resources offered.

We expect our observations on placement and migration to generalize to other applications built on top of location-independent data storage; the commonalities we observed among CoDeeN, Coral, and Bamboo, all of which use request hashing in one form or another to determine where data objects are stored, provide initial evidence to support such an expectation. Given the popularity of content-based routing and storage as organizing principles for emerging wide-area distributed systems, this ap-

plication pattern will likely remain pervasive in the near future. A major class of distributed application generally not built in this way is monitoring applications. A monitoring system could store its data in a hash-based storage system running on a subset of platform nodes, making its behavior similar to the applications we examined in this paper (indeed the SWORD system [15] does exactly that). But another common pattern for these applications is to couple workload to location, storing monitoring data at the node where it is produced and setting up an overlay or direct network connections as needed to route data from nodes of interest to the node that issues a monitoring query [13, 27]. In such systems migration is not feasible. Likewise, data-intensive scientific applications that analyze data collected by a high-bandwidth instrument (e.g., a particle accelerator) may wish to couple processing to the location where the data is produced, in which case migration is not feasible. On the other hand, emerging "data grids" that enable cross-site data sharing and federation may reduce this location dependence for some scientific applications, thereby make computation migration more feasible for data-intensive scientific applications in the future.

PlanetLab is the largest public, shared distributed platform in terms of number of users and sites. Thus, we believe that the platform-specific conclusions we have drawn in this paper can extrapolate to future time-shared distributed platform used for developing and deploying wide-area applications that allow users to deploy their applications on as many nodes as they wish and to freely migrate those application instances when desired. A platform with cost-based or performance-based disincentives to resource consumption would likely result in smaller-scale deployments and more careful resource usage, but *variability* in resource utilization across nodes and over time should persist, in which case the usefulness of matching (and re-matching) application resource demand to node resource availability would too.

On the other hand, our "black-box" view of background platform utilization means our results cannot be easily extrapolated to environments that perform global resource scheduling (e.g., all application deployers submit their jobs to a centralized scheduler that makes deployment and migration decisions in a coordinated way), or in which multiple applications make simultaneous placement and migration decisions. Detailed simulation of platform-wide scheduling policies, and the aggregate behavior that emerges from systems with multiple interacting per-application scheduling policies, are challenging topics for future work. Nonetheless, our analysis methodology represents a starting point for evaluating more complex system models and additional placement and migration strategies. As future PlanetLab-like systems such as GENI [9] come online, and as wide-area Grid systems become more widely used, examining how well these results extrapolate to other environments and application classes will become key research questions.

# 8. REFERENCES


[1] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *OASIS '04*, 2004.

[2] H. Balakrishnan, M. Stemm, S. Seshan, and R. H. Katz. Analyzing stability in wide-area network performance. In *SIGMETRICS*, 1997.

[3] A. Bavier, L. Peterson, M. Wawrzoniak, S. Karlin, T. Spalink, T. Roscoe, D. Culler, B. Chun, and M. Bowman. Operating systems support for planetary-scale network services. In *NSDI*, 2004.

[4] Distributed computing: We come in peace. *Red Herring Magazine http://www.redherring.com/article.aspx?a=10821*.

[5] P. Brett. Iperf. `http://www.planet-lab.org/logs/iperf/`.

[6] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An algebraic approach to practical and scalable overlay network monitoring. In *SIGCOMM*, 2004.

[7] J. A. Crowcroft, S. M. Hand, T. L. Harris, A. J. Herbert, M. A. Parker, and I. A. Pratt. Futuregrid: A program for long-term research into grid systems architecture. In *Proceedings of the UK e-Science All Hands Meeting*, September 2003.

[8] I. Foster et al. The grid2003 production grid: Principles and practice. In *HPDC-13*, 2004.

[9] National Science Foundation. The GENI Initiative. `http://www.nsf.gov/cise/geni/`.

[10] M. J. Freedman, E. Freudenthal, and D. Mazires. Democratizing content publication with coral. In *NSDI*, 2004.

[11] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. Sharp: An architecture for secure resource peering. In *SOSP '03*, 2003.

[12] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *TOCS*, 15(3), 1997.

[13] J. Liang, S. Y. Ko, I. Gupta, and K. Nahrstedt. Mon: On-demand overlays for distributed system management. In *WORLDS*, 2005.

[14] M. Massie, B. Chun, and D. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7), 2004.

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

[16] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughput: A simple model and its empirical validation. In *SIGCOMM*, 1998.

[17] V. S. Pai. `http://codeen.cs.princeton.edu/cotop/`.

[18] V. S. Pai. `http://comon.cs.princeton.edu/`.

[19] S. Rhea, B.-G. Chun, J. Kubiatowicz, and S. Shenker. Fixing the embarrassing slowness of opendht on planetlab. In *WORLDS '05*, 2005.

[20] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: A public dht service and its uses. In *SIGCOMM*, 2005.

[21] M. Ripeanu, M. Bowman, J. Chase, I. Foster, and M. Milenkovic. Globus and planetlab resource management solutions compared. In *HPDC-13*, 2004.

[22] David Spence and Tim Harris. Xenosearch: Distributed resource discovery in the xenoserver open platform. In *Proceedings of HPDC*, 2003.

[23] N. Spring, L. Peterson, A. Bavier, and V. S. Pai. Using planetlab for network research: Myths, realities, and best practices. *ACM SIGOPS Operating Systems Review*, 40(1), 2006.

[24] J. Stribling. All-pairs pings for planetlab. `http://www.pdos.lcs.mit.edu/~strib/pl_app/`.

[25] S. Vazhkudai, J. Schopf, and I. Foster. Predicting the performance

of wide area data transfers. In *IPDPS*, 2002.

[26] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson. Reliability and security in the codeen content distribution network. In *Usenix Annual Technical Conference*, 2004.

[27] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An information plane for networked systems. In *HotNets II*, 2003.

[28] K. Webb, M. Hibler, R. Ricci, A. Clements, and J. Lepreau. Implementing the emulab-planetlab portal: Experience and lessons learned. In *WORLDS '04*, 2004.

[29] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1), 1998.

[30] P. Yalagandula, S. Nath, H. Yu, P. B. Gibbons, and S. Seshan. Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems. In *WORLDS*, 2004.

[31] L. Yang, I. Foster, and J. M. Schopf. Homeostatic and tendency-based cpu load predictions. In *IPDPS*, 2003.

[32] L. Yang, J.M. Schopf, and I. Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Supercomputing 2003*, 2003.

[33] X. Zhang and J. Schopf. Performance analysis of the globus toolkit monitoring and discovery service, mds2. In *Proceedings of the International Workshop on Middleware Performance (MP 2004)*, April 2004.

[34] Y. Zhang, V. Paxson, and S. Shenker. The stationarity of internet path proprerties: routing, loss, and throughput. Technical report, ACIRI, May 2000.