Technical Report

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

TR 07-027

Resource Bundles: Using Aggregation for Statistical Wide-Area Resource Discovery and Allocation

Michael Cardosa and Abhishek Chandra

November 20, 2007

Resource Bundles: Using Aggregation for Statistical Wide-Area Resource Discovery and Allocation

Michael Cardosa and Abhishek Chandra
Department of Computer Science
University of Minnesota
{cardosa, chandra}@cs.umn.edu

Abstract

Resource discovery is an important process for finding suitable nodes that satisfy application requirements in large loosely-coupled distributed systems. Besides inter-node heterogeneity, many of these systems also show high degree of intra-node dynamism, so that selecting nodes based only on their recently observed resource capacities can lead to poor deployment decisions resulting in application failures or migration overheads. However, most existing resource discovery mechanisms rely only on recent observations to achieve scalability in large systems. In this paper, we propose the notion of a resource bundle*—a representative resource usage distribution for a group of nodes with similar resource usage patterns—that employs two complementary techniques to overcome the limitations of existing techniques:* resource usage histograms *to provide statistical guarantees for resource capacities, and* clustering-based resource aggregation *to achieve scalability. Using trace-driven simulations and data analysis of a month-long PlanetLab trace, we show that resource bundles are able to provide high accuracy for statistical resource discovery (up to 56% better precision than using only recent values), while achieving high scalability (up to 55% fewer messages than a non-aggregation algorithm). We also show that resource bundles are ideally suited for identifying group-level characteristics such as finding load hotspots and estimating total group capacity (within 8% of actual values).*

1 Introduction

Recent years have seen increasing use of loosely-coupled distributed platforms for scientific computation [3, 12, 1], data sharing and dissemination [4, 7, 8], and experimental testbeds [5]. For instance, computational grids provide access to large collections of idle computational resources that have enabled large-scale deployments of high-performance and resource-intensive applications [3, 2]. File and content sharing applications [7, 8] exploit the use of spare bandwidth and storage capacity for scalable data dissemination. PlanetLab [5] is another example of a large collection of heterogeneous, dynamic resources that are used for deploying and experimenting with large-scale network applications. In the near future, home appliances and personal appliances are expected to be connected to the Internet [14], providing further opportunities to tap into their computational power for such large-scale cooperative applications.

While such platforms are highly attractive due to their low deployment cost and inherent scalability, they are also highly heterogeneous and dynamic [10]. The nodes participating in such platforms differ widely in their resource capabilities such as CPU speeds, bandwidth, and memory capacity. As a result, resource discovery is often used in such large-scale systems to find suitable nodes that satisfy application requirements. Many existing resource discovery systems [15, 11, 10, 18] rely on the recent resource capacities of individual nodes to make their deployment decisions.

However, resource allocation decisions based on current status of nodes have severe limitations in these systems, because of the presence of intra-node dynamism in addition to the inter-node heterogeneity. Individual nodes can have widely varying resource capabilities due to varying loads, network connectivity, churn, or user behavior. For instance, a resource usage study of PlanetLab [16] has shown that node resource capabilities fluctuate on the order of about 30 minutes. Such dynamism in node-level resource capacities makes it difficult to deploy long-running services and applications that need consistent resource availability to ensure desired performance and avoid disruptions or migration overheads.

To handle the inherent heterogeneity and dynamism in such systems, the resource discovery process employed in such systems must be able to provide *statistical guarantees* on application resource requirements. While incorporating long-term resource availability information is likely to improve the resource discovery decisions substantially [16],

most existing resource discovery systems use only the recent node usage information for scalability and simplicity reasons. It helps in reducing the amount of monitoring data that needs to be exchanged between nodes in the system, and enables easy location of desirable nodes (e.g., by mapping resource requirements to node IDs in case of DHT-based resource discovery systems [15, 11]).

Our contention is that one key reason for this scalability problem with existing resource discovery techniques is the lack of a suitable representation for node resource usage information. In particular, most algorithms try to capture exact recent resource usage of individual nodes, so that it can either be disseminated with low network cost, or be mapped easily to DHT keys. Moreover, most existing resource discovery algorithms try to maintain the resource usage information about individual nodes, which may be redundant since many closely related nodes (e.g., those in the same administrative domain, or running the same application) may have similar usage patterns. We argue in this paper that for providing statistical resource guarantees in a scalable manner, the resource usage information from nodes can be approximated both in temporal (long-term usage pattern) and spatial (number of nodes with similar usage patterns) dimensions.

In this paper, we propose the notion of a *resource bundle*—a representative resource usage distribution for a group of nodes with similar resource usage patterns. This resource bundle employs two complementary techniques to capture the long-term resource usage behavior of a set of nodes: (i) *resource usage histograms* to provide statistical guarantees for resource capacities, and (ii) *clustering-based resource aggregation* to achieve compact representation of a set of similarly-behaving nodes.

Besides providing a scalable resource discovery mechanism to achieve stable application deployment, resource bundles can also be used for several other purposes in a large distributed system. Resource bundles can be used to easily find a *group of nodes* satisfying a common requirement[1]. Resource bundles can also be used to find load *hot spots*: geographical regions in the distributed system with several nodes experiencing overloads due to reasons such as heavy traffic for a popular resource in that region or locality-based application stresses. The identification of such hot spots can be used to inform decisions about application deployment or load balancing. Finally, resource bundles can also be used for *auditing and accounting* purposes, e.g., to determine the resource assignment of a distributed application running on multiple nodes, or to determine the spare capacity in an administrative domain.

We evaluate the performance of resource bundle-based resource discovery using trace-driven simulations and data analysis of a month-long PlanetLab trace. Our results show that resource bundles are able to provide high accuracy for resource discovery through the use of resource usage histograms, (up to 56% better precision than an algorithm based on current usage values), while achieving high scalability through aggregation (up to 55% fewer messages than a non-aggregation algorithm). We also show that resource bundles are ideally suited for identifying group-level characteristics such as finding load hotspots and estimating total group capacity (within 8% of actual values).

[1]SWORD [15] allows the specification of inter-node latencies for multi-node discovery, but there may be other inter-node relationships that may not depend on latencies.

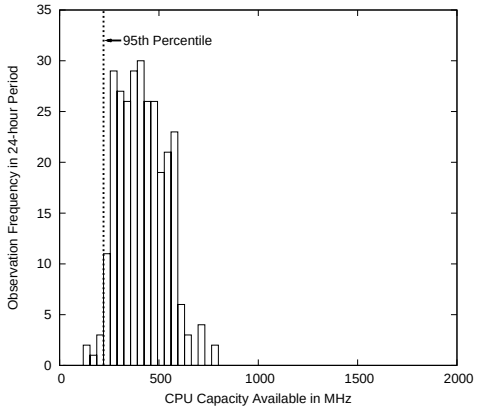


Figure 1. Example of a PlanetLab node's CPU capacity histogram representation, with 95^{th} percentile shown.

2 Statistical Node Behavior

In this section, we present our system model, define a notion of statistical resource usage guarantees, and discuss how historical resource usage patterns can be used to provide these statistical guarantees.

2.1 System Model

We assume our system is a large-scale wide-area distributed system. Participant nodes are geographically distributed and could span multiple administrative domains. We assume the nodes are interconnected by some interconnection overlay, using a DHT or a flooding-based approach, which allows nodes to communicate with other nearby nodes. Nodes monitor their own resource capacities over time and can exchange messages as required. Further, we assume a hierarchical structure can be constructed on top of the overlay, e.g., using methods provided in SDIMS [24].

2.2 Statistical Resource Requirements

During the resource discovery process, applications typically seek nodes meeting certain resource requirements, e.g., minimum CPU spare capacity of 1GHz, memory capacity of 512 MB, etc. Such resource requirements can be expressed as a tuple {R,c}, where R is a resource type, and c is the desired minimum capacity of the resource. However, the resource capacities of nodes in loosely-coupled distributed systems often vary a lot over time, which could result in application performance degradation, failures, or need for frequent migrations [16] resulting in large overhead. It would be desirable to provide statistical resource guarantees so that applications can be deployed on nodes that are likely to satisfy the minimum desired requirement for a certain period of time. The desired length of time could depend on factors such as the overheads of application component migration and restart, or the cost of performance degradation or disruption.

We formalize this notion of statistical resource requirement as follows:

Definition 1 ***Statistical Requirement:*** *We define a statistical requirement r as a tuple {R, c, p, t}, where, R is a resource type, c refers to a capacity level, p is a percentile value, and t is a time duration.*

Intuitively, based on this definition, an application can specify that it needs a resource R to meet a minimum capacity level *c* for at least *p* percent of time (corresponding to its tolerance to overload) over a time duration *t* (which could depend on its length of execution and overheads of distruption and migration, etc.). The goal is to avoid serious service disruptions (e.g. overloads) or reallocation penalties (e.g. migration overheads) over time *t*. Thus, using this definition of statistical requirements, a compute-intensive application can specify a requirement {CPU, 1GHz, 95, 24hrs}, which would mean it requires a 95 percentile CPU capacity of 1 GHz over 24 hours.

2.3 Statistical Resource Usage Representation

Since different applications can specify different values of c, p, and t, maintaining only a few values such as the capacity corresponding to a fixed percentile (e.g., 95th percentile) of resource usage on each node, or the percentile correpsonding to a fixed capacity level (e.g., 1 GHz) may not provide enough information to satisfy different resource discovery queries. Similarly, we may need to capture the resource usage behavior over different time durations (such as an hour, day, week, etc.) to incorporate requirements over different time granularities.

To provide a general way to handle different resource requirement specifications, we propose the use of resource usage histograms with an associated observation time period T, which represent the resource capacity distributions from observations over the past T time units. Figure 1 shows an example of a histogram constructed from a 24-hour time series of CPU capacity for a PlanetLab node, along with the 95^{th} percentile shown (with 95% of the capacity observations to the right of the vertical line). A separate histogram would need to be maintained for each resource type (e.g., CPU, memory, etc.) and for each time granularity (e.g., hour, day, week, etc.); intermediate time granularities can be interpolated from these histograms.

Using histograms to represent resource usage data has two primary advantages:

- Requirement percentiles (corresponding to p) for a particular resource capacity are now straightforward computations from the given histograms.
- Histograms help us preserve all data. Different applications may have different resource requirements and may have different tolerances, expressed as percentiles. We can effectively analyze resource capacity percentiles no matter which requirements and tolerances are given.

Each node can maintain its own histogram by monitoring its own resource capacities over time. Given an arbitrary time period T, nodes can construct histograms from their own historical observations. To save observation overhead, a node may replace its historical trace records with histograms only, using a decay function to have the histograms reflect an approximate trace.

The above technique for statistical resource usage representation through histograms is complementary to any prediction techniques that may be able to predict future resource usage behavior based on historical observations. Predictors are complimentary since histograms merely express usage distributions for nodes; predictors can be used to provide more accurate future estimates of distributions which could then be converted into histograms, enhancing the level of accuracy for future resource capacity estimates.

3 Resource Bundles

While using resource usage histograms provides a means to capture an accurate representation of an individual node's dynamic resource usage pattern and enables the satisfaction of statistical resource requirements, it can potentially create a scalability problem in a large wide-area distributed system. The statistical information for each node would be represented by multiple histograms, corresponding to different resources and different time scales. Disseminating this kind

of detailed data over the network can create significant network traffic. It is infeasible for each node to have a global node-level behavioral view of the entire system. Moreover, if the goal is to find *multiple* nodes meeting a certain requirement, it would be desirable to combine this discovery process rather than having to find individual suitable nodes separately.

Thus, the question is whether we can use these representations in a scalable manner to make better resource discovery descisions in a large system? Secondly, can we use these node behaviors to provide any collective information about group-level usage patterns, e.g., for nodes within an administrative domain, or for nodes assigned to a distributed application?

3.1 Resource Aggregation

Aggregation [24], particularly hierarchical aggregation [6], is a common technique employed in large distributed systems for the scalable dissemination of information. Aggregation essentially compresses the amount of transmitted data in the system while preserving the overall information content. In the context of resource discovery, this would correspond to a suitable "compression" of the node resource usage patterns to achieve a desirable tradeoff between the quality of resource discovery and the overhead of network data transmission in the system.

In other words, our goal is to achieve the same quality of resource discovery as a global resource discovery system with full historical node-behavioral knowledge, but to significantly compress the amount of necessary node-behavioral representation data in the system in order to achieve scalability and to produce precise, timely results. Such an aggregation of node resource usage distributions for a group of nodes can be used to represent:

- An accurate approximation of any individual node's resource usage in the group. Such an approximation is important for the discovery of desirable nodes based on a resource requirement.
- The collective resource usage behavior of the group of nodes. This can provide information about load patterns or dynamic resource usage behavior for a set of related nodes (e.g., those in the same geographical area or running the same application).
- The overall capacity of the group. This can be used to track the overall resource usage, e.g., for audit or accounting purposes.

A naive approach to aggregation for a set of nodes would be to compute the average resource capacity distribution across all nodes. An example of this naive approach can be seen in Figure 2. While this averaging allows the estimation of overall capacity of the group of nodes, it is a poor representation of individual node-level behavior. For instance, as shown in Figure 2, while one node is skewed towards low capacity and another towards high capacity, the average representation loses this information and appears to represent a set of bimodal nodes.

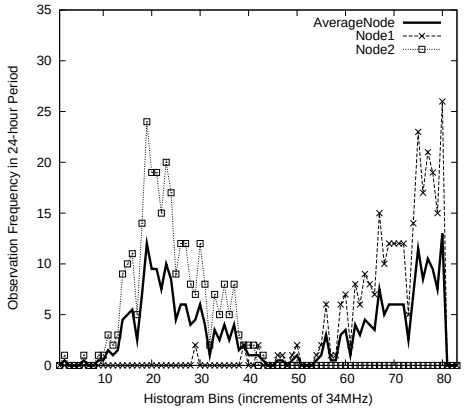


Figure 2. Naive approach to aggregation by averaging resource capacity distributions.

The problem with this naive approach is that it does not account for the heterogeneity of the nodes in the system, and fails to capture important behavioral differences between individual nodes. Thus, this approach could result in the combination of nodes that have widely different resource capacity distributions, thus providing a highly inaccurate view of individual nodes' resource capacities.

3.2 Resource Bundles: Aggregation via Clustering

To account for the heterogeneity of nodes, we define the notion of *resource bundles:* an aggregation of a group of nodes with *similar* resource capacity distributions. An average distribution could be computed over a resource bundle to achieve a compact representation of the individual node resource capacity distributions. We refer to such an average distribution as a *bundle representative*. Such an aggregation process will preserve the individual node distributions more accurately, while bundle representatives can be used to provide group-level resource capacity information.

Figure 3 shows a high-level view of the notion of resource bundles and how they might be constructed. First, a group of nodes are bundled based on the similarity of their resource capacity histograms. Second, each bundle produces a representative distribution that can be used to characterize the whole bundle. But how can we identify

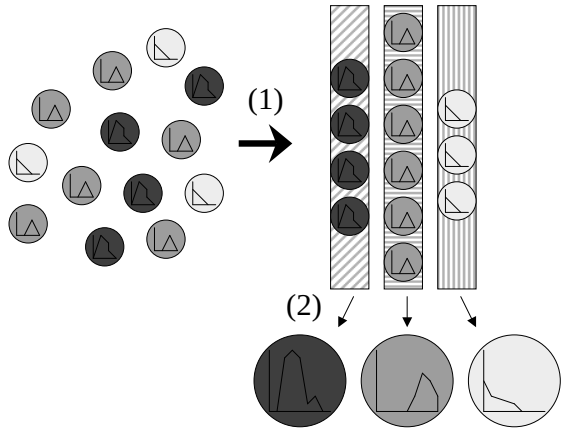


Figure 3. High-level view of the clustering process. First, nodes are grouped by similar behavior distributions (1), then bundle representatives are formed for each bundle (2).

such groups of similar nodes to construct a resource bundle, and how can we compute the bundle representative to accurately represent the nodes in the bundle?

To identify nodes with similar distributions, we propose the use of *clustering* algorithms that have traditionally been used in data mining applications to group together statistically similar data elements. However, a clustering algorithm must meet several requirements in our context:

- The data to be clustered (i.e., the node resource histograms) is not single-point, but multi-element (consisting of multiple histogram bins). The clustering algorithm must be able to *handle such multi-element data*.
- The node resource usage histograms could represent arbitrary distributions, and cannot be assumed to conform to standard distributions (e.g., Gaussian, uniform, etc.). The clustering algorithm must be *distribution-free*, i.e., it must not assume the existence of a standard distribution or certain parameters.
- The clustering algorithm must not only identify the closely related set of nodes, but it is desirable if it can also produce a *compact representation of the collective resource usage* of these nodes. Such a representation can be used to easily characterize the nodes in a bundle (e.g., high-capacity/low-capacity, etc.)

A clustering algorithm that meets the above requirements is the *multinomial model-based expectation maximization (EM)* clustering algorithm [26]. This clustering algorithm has been used primarily for the purposes of clustering text documents with common words. We first describe this algorithm in a document clustering context for ease of exposition, and then describe how it maps to our context.

In a document clustering context, each document is considered as a "bag of words", and is represented as a vector of word frequencies based on how many times each word appears in the document. Then, the set of all documents is represented as a mixture of multimonial distributions on these word frequencies, with each document belonging to one such distribution. The probability that a document belongs to one of the clusters corresponding to a multinomial distribution is given by [26]:

$$P(d_i|\lambda_j) = \prod_l P_j(w_l)^{n_{il}}, \quad (1)$$

where λ_j is the set of parameters for model j, n_{il} is the frequency of occurrences of word w_l in document d_i, and $P_j(w_l)$ is the probability of word w_l occurring in cluster j. Further, $\sum_l P_j(w_l) = 1$ holds.

Mapping the document clustering scenario to our context, we can think of nodes corresponding to documents and histogram bin magnitudes for node-level resource usage distributions corresponding to document word frequencies. The clustering algorithm then maps nodes to clusters based on the similarity of their resource usage histograms. In other words, this algorithm will group those nodes together that have similar histogram bin-magnitudes, meaning it can cluster nodes having arbitrary (but similar) distributions. Such a cluster of closely related nodes returned by the clustering algorithm is considered a resource bundle.

In practice, the multinomial model-based EM clustering takes a set of vectors as its input and forms clusters based on the similarity of corresponding vector elements. It is a hill-climbing algorithm that adjusts its mapping of vectors to clusters iteratively in order to maximize the expected objective value achieved from its clustering.

Notice that this clustering algorithm meets the requirements we had outlined above. The algorithm is able to handle multi-point data as it operates on vectors of values. Since the clustering algorithm operates on arbitrary vectors, it is independent of any assumptions about specific distributions, enabling the clustering of arbitrary resource capacity distributions. Finally, by associating each cluster to a multimonial distribution, it is able to characterize the common statistical properties of the nodes in a cluster in a compact manner.

3.3 Bundle Representatives: Aggregate Distributions

As described above, the multinomial model-based EM algorithm associates each resource bundle with a multinomial distribution that captures the statistical properties of

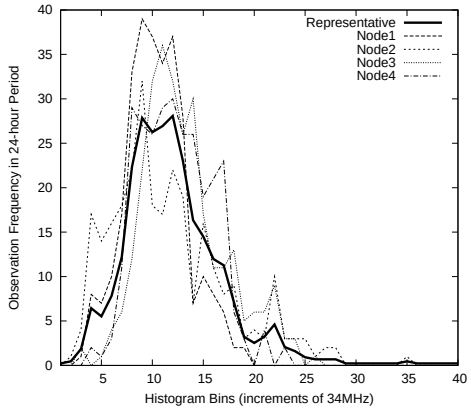


Figure 4. An example of a CPU capacity clustering of 4 nodes, producing a *bundle representative*

the nodes that are members of the bundle. This multinomial distribution can be used as the bundle representative: a compact representation of the individual nodes distributions in the bundle.

Bundle representatives are the resulting distributions that represent the mean node capacity distribution within each respective bundle. In the multinomial model clustering algorithm, cluster representatives are the result of probabilistic models coupled with Laplace smoothing; however the difference between these models and the mean node capacity distribution is negligible.

Figure 4 shows an example bundle representative that aggregates 4 node resource capacity histograms. As seen from the figure, the representative closely matches the individual node distributions. In Section 4, we will quantify the closeness of the representative to its member distributions, and its effectiveness in resource discovery, hotspot detection and capacity estimation.

3.4 Hierarchical Aggregation

While aggregation, as described above, enables the creation of resource bundles that closely approximate individual node behavior for a similar set of nodes, the question is whether bundles can be further aggregated to provide a meaningful view of the combined set of nodes corresponding to multiple bundles. The ability to combine resource bundles is particulary desirable in a large system where we may want to get concise estimates of resource capacities of nodes at different granularities; for instance, at a local site level, at an administrative domain level or at a global level.

Building on the assumed ability of the system-employed overlay to support a hierarchical information structure, we

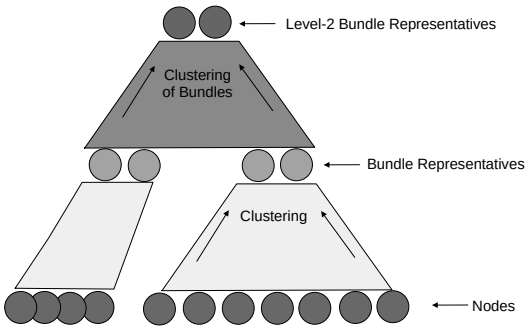


Figure 5. Hierarchical aggregation

propose the use of *hierarchical aggregation* [6] in combination with resource bundles through the use of recursive clustering, i.e., successive clustering of bundle representatives at different levels of the hierarchy. Figure 5 is a high-level illustration of this process. Groups at the bottom level are individual sets of node distributions. These nodes are initially clustered, producing bundle representatives which are then propagated to the next level of the hierarchy to create level-2 representatives, thus beginning the recursive represenative clustering process.

Note that this recursive aggregation must incorporate individual bundle cardinalities during the clustering process to minimize the loss of representative data. In an instance of recursive clustering, representatives with highly different cardinalities might be combined. If the bundle representatives are all treated alike, low cardinality representatives might have an unusually large influence in the formation of higher-level representatives. For example, if a bundle representative of 1000 member nodes is combined with another representative of cardinality 10, their respective contributions to the resulting aggregate should be proportional to their respective cardinalities, so that smaller cardinality representatives do not have undue influence.

To balance the relative importance of different resource bundles into the resulting aggregate bundle, we use bundle cardinalities as *representative weights* in the process of hierarchical clustering. Formally, for a set of representative resource capacity histograms $H = H_1, H_2, ..., H_n$, with each respective histogram H_i having cardinality w_i, and after presenting each histogram to the clustering algorithm as $W_i = H_i w_i$, we define the combination of the alike histograms in H into a bundle representative histogram C as

$$C = \frac{1 + \sum_{i=1}^{n} W_i}{\sum_{i=1}^{n} w_i} \quad (2)$$

The resulting bundle representatives[2] at any level in the hierarchy can thus be used for any arbitrary capacity re-

[2]The addition of 1 to the numerator in Equation 2 effectively adds 1 to each histogram bin. This is important for the probabilistic model in the algorithm; no histogram bin should have a probability of 0.

quirement; the resource discovery system need not know capacity requirements apriori. Rather, the bundle representatives are adequate representations of their respective descendants.

If the underlying topology of the distributed system causes nodes to be grouped by locality, our hierarchical approach will be suitable for finding localized sets of resources for application deployment.

Further, bundle representatives can be used to predict group-level capacity levels. Since similar node-level capacity distributions are clustered together, our hypothesis is the variance of bundle-level capacity distributions will be low. Our ability to predict group-level capacities results in another high-level ability to detect hotspots within large distributed systems.

4 Evaluation

We now evaluate the performance of using resource bundles for resource discovery, capacity estimation and hotspot detection using data analysis of a month-long PlanetLab trace as well as a trace-driven simulation. We begin with our data analysis results and present the simulation results in detail in Section 4.8.

4.1 Data Analysis Methodology

We used a PlanetLab trace obtained by CoMon [17] from February 2007 for our experiments. CoMon collects node-level resource capacity information every 5 minutes. We used *free CPU capacity* as the resource of interest for our evaluation, which was estimated using the *CPU Burp* statistic: it is calculated by occasionally running a spin-loop to determine how much CPU bandwidth could be obtained by a newly-deployed application. We multiply the CPU Burp by the CPU speed to determine the total amount of node-level free CPU. During February 2007, 427 PlanetLab nodes contributed at least one data point per day and thus were chosen for our analysis. We used a time period of 24 hours as the desired time window, thus computing node histograms on a 24-hour period, one per day. Each histogram consisted of 100 bins, each one representing about 34 MHz, with 3.4 GHz assumed as the maximum CPU capacity in PlanetLab, as well as a 101st bin for any higher observed capacities.

We used MatLab as our main tool for data analysis incorporating an already implemented Matlab package for the multinomial model EM clustering algorithm [25]. Since this algorithm is hill-climbing, we ran it with 100 random initializations taking the best objective value to determine the best clustering. We emulated single-level (flat) clusterings, as well as hierarchical clusterings with 2 and 5 levels over the trace.

4.1.1 Emulating Resource Discovery

To evaluate the accuracy of a resource bundle-based resource discovery process in finding desirable nodes, we emulate a resource discovery process as follows. We run a resource discovery algorithm on an *observation time window* to determine the choice of acceptable nodes that meet a desired resource requirement. We then compare this choice of *acceptable nodes* to the actual set of nodes that satisfied the desired requirement over a *solution time window*: the time window in the trace during which these nodes would have been allocated to the application.

For the purposes of our experiments, we defined the statistical requirement r={CPU, c, p, 24hrs}, with different values of c and p. The goal of the resource discovery algorithms was to find all nodes meeting requirement *r* on the 427-node PlanetLab trace. Notice we did not specify how many nodes an application needs for its deployment. Instead of using this as a parameter in our initial analysis, we had the algorithms search for the complete set of acceptable nodes.

In our experiments, we use the entire February 2007 trace by using each day's trace (starting from Feb 1 data) as the observation window with the next day's trace being used as the solution window. The observation and solution windows are then shifted by one day, thus giving us 27 samples of trace data to evaluate our algorithms.

4.1.2 Comparison Algorithms

We used the following resource discovery algorithms for comparison:

- ***Memoryless***: This algorithm uses the last CPU capacity data point for each node to estimate its expected capacity over the next day. This algorithm emulates resource discovery algorithms that use recent resource usage information to determine the suitability of a node to meet a minimum requirement, and does not incorporate statistical resource usage patterns into its decisions.

- ***History***: This is a centralized algorithm with global historical knowledge of the entire system. It maintains complete 24-hour CPU capacity histograms for each node. Each node histogram is individually examined to determine which nodes meet the desired statistical requirement r, thereby determining the set of acceptable nodes. This algorithm can be considered to be the best we can do using historical observations without using any accurate predictors. It also provides us with a baseline to determine the effect of data loss due to aggregation on the accuracy of resource discovery.

- ***Aggregation***: This algorithm uses resource bundles to aggregate the resource usage histograms of groups of

nodes into resource bundles. Nodes are bundled into k bundles based on histogram similarity. Bundle representatives are then used to determine the set of acceptable nodes: for each bundle, if its representative meets the desired statistical requirement r, all of its members are selected as acceptable nodes.

4.1.3 Evaluation Metrics

The accuracy of resource discovery was measured using two statistical metrics: precision and recall, defined as follows:

$$Precision = \frac{n_{acc}}{n_{tot}} \quad (3)$$

and

$$Recall = \frac{n_{acc}}{N_{acc}} \quad (4)$$

Here, n_{acc} = acceptable nodes chosen, n_{tot} = total nodes chosen, N_{acc} = total acceptable nodes in the system.

Intuitively, high precision means that a high fraction of the nodes returned by a resource discovery algorithm are actually acceptable, thus reducing the chances of poor allocation decisions. On the other hand, recall measures what percentage of the total acceptable nodes in the system are discovered by the algorithm, indicating how well a resource discovery algorithm can locate acceptable nodes in the system. Note, however, that if an algorithm exhibits a high level of recall, this says nothing about the overall quality of its set of chosen nodes.

Essentially, these measures compare resource discovery algorithms against an "oracle" that has perfect knowledge of future behavior, whose precision and recall are both 1.

4.2 Accuracy of Aggregation for Resource Discovery

Figure 6 compares the accuracy of our single-level aggregation algorithm using 10 clusters[3] against the memoryless and history-based resource discovery algorithms. This figure shows the precision and recall achieved by the algorithms for three different CPU capacity requirements as well as the *mean requirement* (MeanCPU), the average measure over all possible requirements (one for each of the 100 histogram bins).

As seen from Figure 6(a), both the Cluster and History algorithms have significantly better precision (87-97%) than the Memoryless algorithm (37-77%). This results from their use of more historical information to make more statistically accurate decisions. The algorithms with historical capacity distributions select a node only if it has had a resource capacity of at least c CPU MHz for at least $p = 95\%$ of the observations in the past 24 hours. On the other hand, Memoryless will select a node if it has a capacity of c MHz at its last observation point, which is a much less stringent requirement.

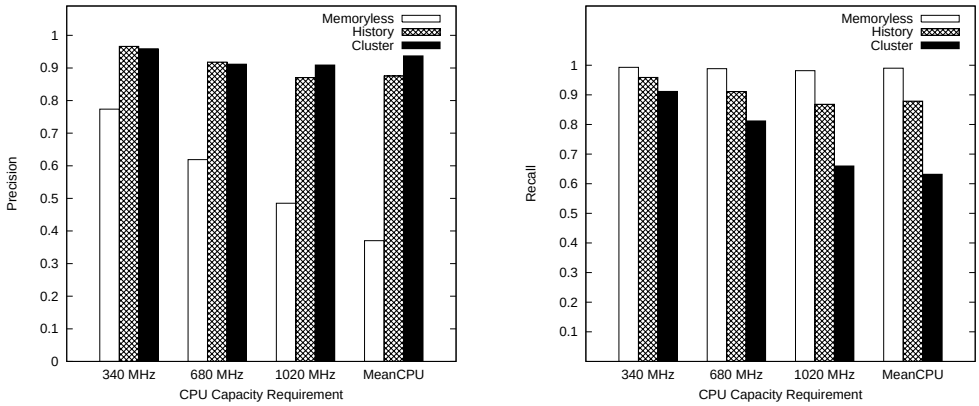
This same behavior also explains the recall values in Figure 6(b), which shows Memoryless had the highest recall, while History came in second, and Cluster consistently had the worst recall. The Memoryless algorithm turns out to be an unusually optimistic predictor; it finds nearly all of the acceptable nodes but also finds many other unacceptable (but temporarily well-performing) nodes as well, leading to poor precision but excellent recall. This result can be understood by noting that the Memoryless algorithm would find, on average, $p = 95\%$ of the acceptable nodes in the system along with every other node that had a resource capacity $c_t \geq c$ at the moment of observation t. The Cluster algorithm suffers in recall because it is conservative similar to History. However it misses additional acceptable nodes due to the loss of accuracy because of aggregation, that might sometimes combine acceptable nodes along with unacceptable nodes in the same bundle. Thus it achieves a higher precision than History but experiences worse recall.

However, when we look at the absolute number of nodes discovered by each algorithm in Figure 7, we find that the actual number of nodes missed by Cluster is typically between 10-20, even in the worst recall case (MeanCPU - 13 nodes missed), while the number of additional (unacceptable) nodes returned by Memoryless can be in the order of 50-85 nodes. This indicates that the impact of missing acceptable nodes by Cluster is comparatively smaller than that of finding additional poor nodes by Memoryless.

To consider the relative impact of high precision vs. low recall and vice versa, we note that the *goodness of choice* of nodes from the application's perspective is primarily affected by precision. Once the allocation decision has been made, precision ultimately reflects the confidence of the application in the selected nodes meeting the given requirement. On the other hand, recall's effects are dependent on the system context. In a system where very few nodes meet the given requirement, a high recall may be desired so that most of the acceptable nodes could be found. However, in a large system with several acceptable nodes, recall would primarily affect query latency: low recall implies that acceptable nodes will be missed, therefore taking the query longer to find the desired number of acceptable nodes. But poor precision and good recall may adversely impact the performance of an application if a choice of poor nodes affects the execution of the application after deployment.

In summary, our precision and recall evaluations show that resource discovery using *our resource bundle approach*, even though it experiences data loss through aggregation, *is able to find a choice of nodes similar in quality to those discovered by a fully-informed history-based algo-*

[3]We investigate the impact of number of clusters on aggregation accuracy in Section 4.6.



(a) Precision (b) Recall

Figure 6. Aggregation (Cluster) compared against baseline resource discovery algorithms (95^{th} percentile).

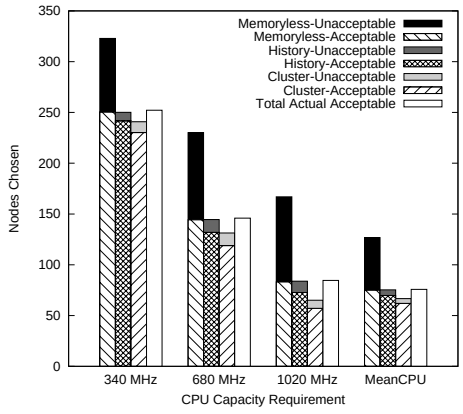


Figure 7. Number of nodes chosen as acceptable nodes for the same experiment as Figure 6

rithm, although its set of chosen nodes is smaller, thereby missing a few potentially acceptable nodes.

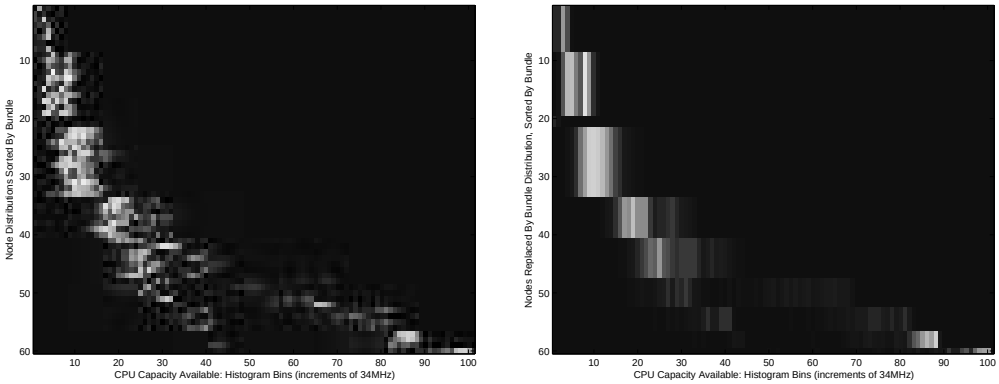
4.3 Using Aggregation to Identify Group-level Characteristics

One potential advantage of using resource bundles is to concisely capture group characteristics of sets of nodes such as their overall load behavior and spare capacity. Such group-level characteristics could inform decisions of load balancing or capacity planning without having to rely on fine-grained node-level statistics. Next, we perform an experiment to evaluate this potential of resource bundles for geographically related nodes in PlanetLab.

To establish proximity-based groupings of nodes for our experiment, we hand-selected 5 geographically distributed group leaders in our PlanetLab trace. These leaders were respectively from UMASS Amherst, UFL, UT Austin, UWash, and UMN, representing different US regions. Pairwise pings between these leaders and the remaining PlanetLab nodes were taken, forming a total of 300 responsive PlanetLab nodes. We then formed 5 groups of 60 nodes each based on their proximity from the selected group leaders.

4.3.1 Load Hotspot Detection

Figure 8(a) shows a visual display of the resource usage histograms of 60 nodes from one of the geographical groups. The figure is a gray-scale image where the x-axis corresponds to CPU capacity, the y-axis corresponds to node identifiers (sorted based on the bundle identifier assigned by our clustering algorithm), and the brightness of a point corresponds to the magnitude of the histogram bin. Thus, a node with brighter values towards smaller CPU capacity values (e.g., node 10) has a smaller spare CPU capacity available. Figure 8(b) shows the corresponding bundles representatives we obtain by our clustering algorithm. As can be visually seen from this figure, low capacity nodes are clearly separated from high capacity nodes, and further we can easily estimate the number of nodes in each category simply by the cardinality of the corresponding bundle. For instance, cluster 2 (consisting of nodes 9 to 19) appears to have very low spare CPU capacity, and could potentitally be overloaded. Using such group-level views can allow administrators to identify potential load hot spots by using ap-



(a) Nodes, sorted by bundle rep (b) Nodes, replaced and sorted by bundle rep

Figure 8. Distributions of a local group of nodes sorted by bundle. Bundle reps with their respective weights can be used to detect hotspots in the network.

propriate thresholds.

4.3.2 Capacity Estimation

We now investigate the capacity estimation abilities of our aggregation algorithm. Here, we ask the following question: To what level of accuracy can the aggregation-based algorithm estimate the expected resource capacity of a group of nodes for the next day based on observations of a 24-hour period? We compare the results to those of a history-based capacity estimation algorithm that has fine-grained knowledge of individual node resource usage. For the aggregation-based algorithm, the group capacities are estimated by taking a weighted sum of the mean capacities of the bundle representatives, while the mean capacities of individual nodes are added up for the history-based estimation algorithm.

Figure 9 shows our results. As seen from the figure, using 10 clusters, the aggregation-based algorithm is able to estimate within 3% accuracy of a history-based estimation algorithm. To see how sensitive aggregation is to the choice of a good cluster size, we also include an average of our group-level predictions for all clusterings that used between 5 and 20 clusters (the third bar in the figure). The results show that the impact of the cluster size is minimal ($< 1\%$ accuracy).

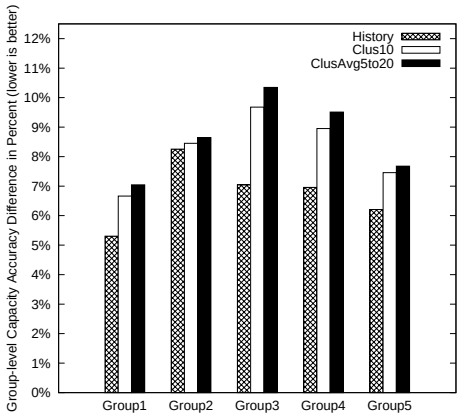
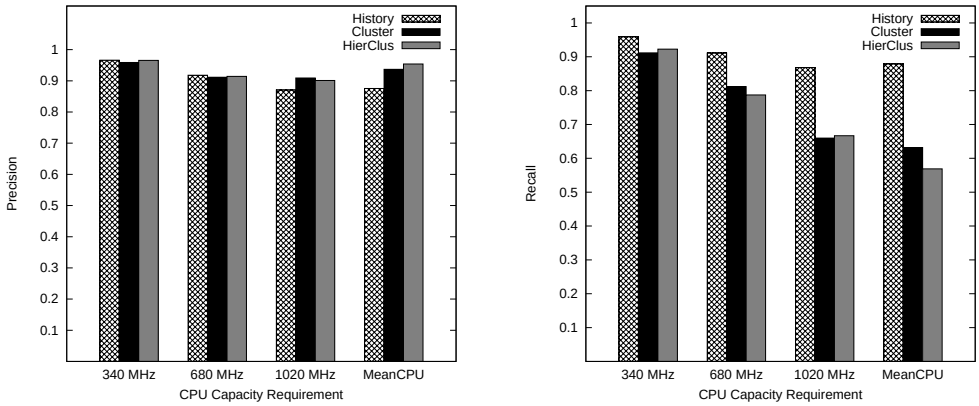


Figure 9. Accuracy of group-level capacity estimations using Aggregation (Clus10) and a fully-informed (History) algorithm.

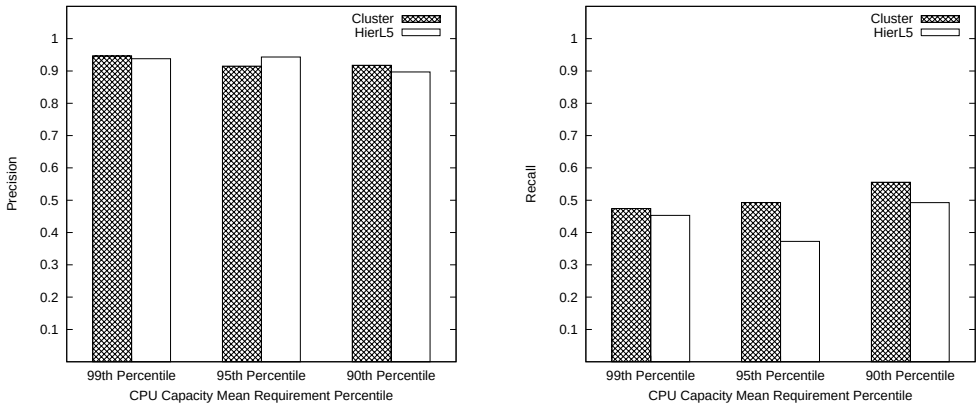
4.4 Impact of Hierarchical Aggregation

We now examine the impact of hierarchical aggregation on the accuracy of our algorithm. As described in Section 3.4, hierarchical aggregation corresponds to recursive bundling, with the goal of achieving scalability in a large



(a) Precision (b) Recall

Figure 10. A 2-level hierarchical clustering algorithm has slightly better precision but worse recall than a flat-clustering algorithm. (Results for groupsize $y = 70$ nodes per group, and $p = 95^{th}$ percentile.)



(a) Precision (b) Recall

Figure 11. A 5-level hierarchical aggregation algorithm (HierL5) compared against baseline single-level aggregation (Cluster). Results are shown over requirement percentiles.

distributed system. Here, we examine the impact of this further approximation of node-level capacity.

4.4.1 2-Level Hierarchy

We first examine a 2-level hierarchy that consists of 420 PlanetLab nodes divided into 6 random groups of size 70. The hierarchical aggregation algorithm (HierClus) works as follows: It first reduces each group of nodes to 10 resource bundles. The respective 10 resource bundles from each group, 60 in all, are then further aggregated into 10 second-level resource bundles, representing the aggregate resource usage behavior of the 420 nodes. For resource discovery purposes, the algorithm examines the 10 second-level bundle representatives and if a bundle representative H_i meets requirement r, then all nodes represented by H_i (recursively) are chosen by the algorithm.

Figure 10 compares HierClus with Cluster for precision and recall statistics. Precision does not suffer from the recursive aggregation in HierClus, while the recall is slightly lower (about 6% for MeanCPU). This shows that the loss of accuracy in hierarchical clustering is small compared to a single-level clustering.

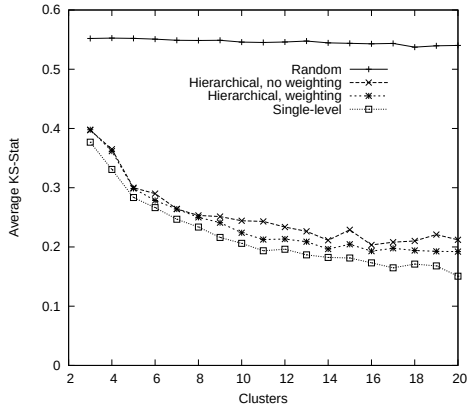


Figure 12. Data loss measured by the K-S Statistic which is a measure of the average difference between a node's distribution and that of its bundle representative. Results are shown over the numbers of clusters used in the clustering algorithms Random, Cluster, and HierClus.

4.4.2 5-level Hierarchy

Since 420 nodes is small to emulate a larger hierarchy, we emulated a large distributed system by using PlanetLab traces from each day in our one-month trace to represent a different node (two consecutive days' traces were still used for observation and solution time windows). This provided us with $427*27 = 11,529$ node distributions, each of which was assigned randomly to one of 15 different locations in the system. We then constructed a 5-level hierarchy over this system using group sizes of 70 nodes each, and 10 bundles per group at each level.

We evaluated this 5-level hierarchical aggregation algorithm (HierL5) against a single-level aggregation algorithm (Cluster) in the same fashion as the 2-level hierarchical evaluations. The results are displayed as different percentiles for requirements. Also for these results, we observed the mean requirement starting from a 34MHz requirement up to a requirement of 1.36GHz. Figure 11 shows the precision and recall of HierL5 as compared to the single-level aggregation algorithm. Once again, we see that the precision of hierarchical aggregation is very close to the single-level aggregation, with a small loss in recall.

4.5 Aggregation Information Loss

While aggregation helps in the scalability of resource discovery and the capturing of group-level characteristics by reducing the amount of resource usage data to be maintained and examined, we next examine how much information loss occurs as a result of this reduction in data.

We measure this information loss by comparing each node-level resource distribution to its bundle representative. We use the Kolmogorov-Smirnov (K-S) statistic D_{node} for each node as a quantitative measure for this comparison:

$$D_{node} = \sup_x |F_{node}(x) - F_{rep}(x)|,$$

where F_{node} and F_{rep} are CDFs for the resource capacity distributions of the node and its bundle representative respectively. The K-S statistic is a value between 0 and 1 indicating the magnitude of difference between the two distributions. The more similar the two distributions are, the closer is the K-S statistic is to 0. The K-S statistic captures the information loss based on the overall goal of aggregation: to have the bundle representatives be as similar to their represented nodes as possible.

For our evaluation, we compare the K-S statistic for the following algorithms: single-level aggregation, hierarchical 2-level aggregation, hierarchical 2-level unweighted aggregaton (where we do not consider bundle cardinalities during recursive aggregation), and random (which assigns nodes randomly to equal-sized bundles). Here we use the random algorithm as a baseline for comparison. Note that a history-based algorithm would have a K-S statistic of 0, since it uses individual node-level distributions.

Figure 12 shows the average K-S statistics for these algorithms. The K-S statistic has a worst-case value of about 0.38 for the single-level aggregation, but reduces to about 0.21 for a cluster size of 10 (which is the value used in our previous experiments), compared to a value of 0.55 for the random algorithm. While a K-S stat value of 0.21 may seem a bit high (considering a cutoff value of 0.05 typically used for statistical significance tests), this result still shows that the loss of information is relatively low considering that 420 distributions have been reduced to 10 distributions. Also, the results from previous subsections have shown that this information loss has only a small impact on the accuracy of resource discovery and capacity estimation.

We also see from the figure that hierarchical aggregation results in a small loss of information, and that using bundle cardinalities as weights provides more accurate results compared to the unweighted algorithm.

Another observation from the figure is that the value of K-S statistic decreases with increasing number of clusters. This is as expected, because as the number of clusters increases, each cluster has fewer constituent nodes, resulting in more accurate aggregations. In the limit, having as many clusters as the number of nodes would result in a K-S stat close to 0.

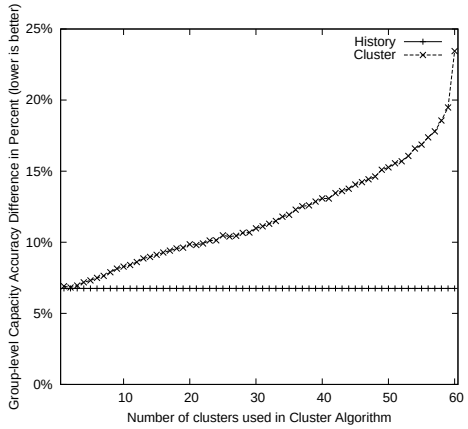


Figure 13. For group-level predictions, fewer clusters showed better results.

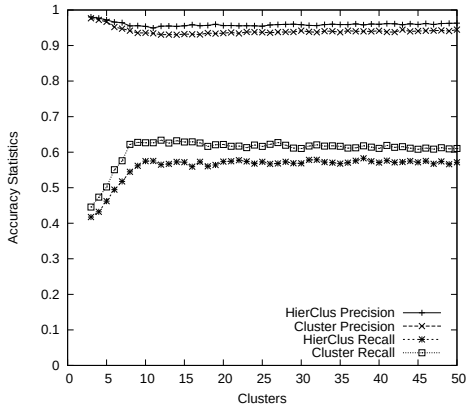


Figure 14. Precision and recall for varying numbers of clusters in the clustering algorithm, for the mean CPU requirement ($p = 95\%$). This shows a value of 10 clusters is appropriate for this system.

4.6 Effect of Clustering Parameters

We next examine the effect of clustering parameters on the accuracy of aggregation. In particular, we explore the impact of the number of clusters used by the clustering algorithm as a parameter to generate the desired set of resource bundles.

Figure 13 shows the accuracy of capacity estimation. The figure shows that the accuracy decreases as we increase the number of clusters. This result seems counter-intuitive as one might expect the accuracy to converge to that of History when the number of clusters approaches the number of nodes, as was observed for information loss in Figure 12. However, this inaccuracy occurs because the multinomial model EM clustering algorithm uses a probabilistic model to generate bundle representatives. As a result, the representative distributions are not identical to the individual node distributions in the limit, and there is always a small amount of difference in the resulting distribution[4]. With a high number of clusters, this difference gets amplified due to the additive nature of capacity estimation.

Together, the results from Figures 12 and 13 show that while having too few clusters might result in inaccurate bundle-level aggregation, having too many clusters can adversely impact group-level estimates. This implies that some intermediate values might be desirable to achieve a good balance between the two requirements. This result is also evident from Figure 14 that shows the accuracy of resource discovery over the number of clusters used in our clustering algorithm. The figure shows diminishing returns in accuracy occuring beyond the knee of the curve, which occurs at $numclusters = 10$. In general, this value is dependent on the dynamism and heterogeneity of the system.

[4]Note that this difference is expected as the probabilistic model is merely using the observed distribution as a set of samples that are being fitted to a general distribution from which these samples are drawn.

4.7 Observation Time Window length effects

So far we have been using a 24-hour observation time window for our 24-hour test set (prediction period). We wanted to explore the effects of varying the size of this observation window to see if we could tune the balance between precision and recall for our HierClus algorithm. We also wanted to see what would happen to the baseline History algorithm under the same circumstances.

Figure 15 shows the results. The most interesting results occur at the 24-hour mark. For both algorithms, precision plateaus and recall diminishes after the 24-hour mark. This can be explained by an increasing amount of stale data being used to make relatively shorter and shorter time-period predictions.

As the observation window decreases from 24 hours down to 1 hour, we see the precision of HierClus increase while the recall of HierClus decreases; hence, HierClus is choosing fewer nodes, but the nodes it chooses are still highly acceptable. This is an additional side-effect of poorer clusterings as less data is received; clustering performance is poor with insufficient data.

The results indicate that a 24-hour observation window is the best size for a 24-hour prediction period. If less than 24 hours of observation data is used, then the clusterings are poorer and recall is worse. If more than 24 hours of observation data are used, data staleness becomes an issue,

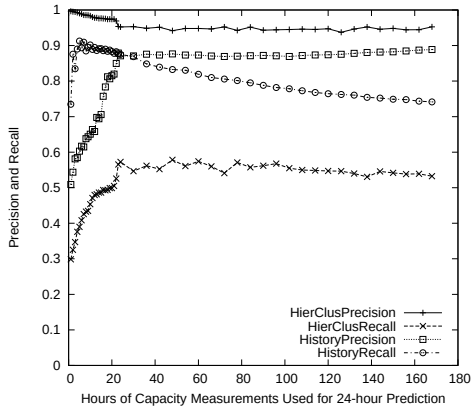


Figure 15. Precision and recall as a function of the observation time window length for 24-hour predictions. Recall diminishes and precision plateaus after the 24-hour mark.

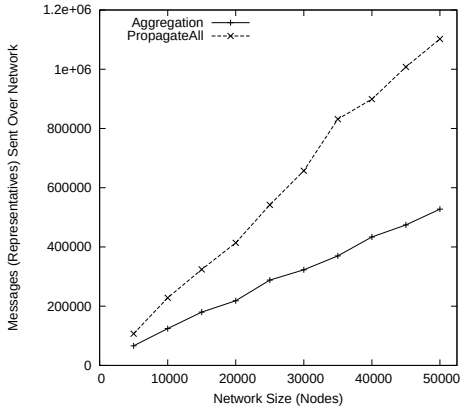


Figure 16. Aggregation overhead increases at a slower pace than PropagateAll with the same hierarchical toplogy.

causing diminishing recall but steady precision.

Precision is steady due to the fact that if nodes meet the given requirement for 95% of the observations for a very large observation window, that they are highly likely to show the same resource capacities in the future.

4.8 Trace-Driven Hierarchical Aggregation Simulation

We now present results from a trace-driven simulation of a hierarchical aggregation algorithm. The purpose of these simulations was to quantify the overhead and query latency of our resource discovery algorithms in a large-scale system with a realisitic topology. We have based our simulator on PeerSim, a widely used simulator for distributed algorithm evaluation. We have used our month-long PlanetLab trace to drive these simulations.

4.8.1 PeerSim Methodology

A hierarchical overlay was implemented using *WireInet-Topology* in PeerSim. The depth of the hierarchy varied from 5 to 7 levels. Since single-node distributions may now be bundled together with multi-level bundles, in this context we now define a *bundle* as a representation of 1 or more nodes.

At each simulator cycle, each node propagates its bundle list upwards to its parent. Between each cycle, each node, upon receiving its messages, updates its data store and if the number of bundles it stores exceeds a *maximum bundle threshold* B_t, the aggregation algorithm is executed at that node, consolidating its current list of bundles into B_a aggregate bundles.

We applied our clustering aggregation algorithm into this topology. We also applied a baseline ***PropagateAll*** algorithm into this same topology with threshold $B_t = \infty$, where no aggregation takes place; all node-level capacity distributions are propagated all the way to the root. Another baseline algorithm we used was ***PropagateNone***, where no resource discovery information is propagated between nodes beforehand; resource discovery must be performed through the probing of each node individually for its capacity distribution. All three algorithms use bundle histograms for resource discovery purposes.

4.8.2 Overhead

We investigated the overhead of Aggregation and PropagateAll by measuring the data transfer necessary for a complete system-wide propagation to the root node in the heirarchy. Data transfer is measured in the number of bundles sent over the network. For Aggregation, we used $B_t = 70, B_a = 10$. Figure 16 measures overhead by the size of the network. As seen in the figure, the overhead of Aggregation increases at a slower pace than PropagateAll. We see no asymptotic difference between Aggregation and PropagateAll since they operate on the same hierarchical topology structure.

4.8.3 Query latency

We now analyze query latency from our trace of 11,529 PlanetLab node distributions as in Section 4.4.2. We chose not to fix an application-desired number of acceptable nodes

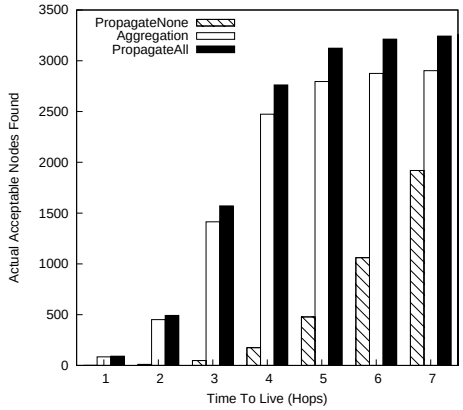


Figure 17. Aggregation finds nearly as many nodes as PropagateAll, but with half the data transfer overhead in Figure 16. PropagateNone rebounds in the end, but with significant query-time overhead.

to be found; instead, we measured how many actual acceptable nodes were found by the resource discovery algorithms given a time-to-live value measured in hops. Using the node-level requirement {CPU, 680MHz, 95%, 24hrs}, we determined the number of actual acceptable nodes found by the average of 500 samples in the system. Bundles were chosen based on their histogram as in previous sections; the number of actual acceptable nodes found was determined by evaluating which of the selected nodes actually met the given requirement over the following day.

Figure 17 shows the results. Aggregation finds nearly as many nodes as PropagateAll, however with about half the data transfer overhead in the same hierarchical structure. PropagateNone starts to rebound at 7 hops, but with significant query-time overhead, as it must query each individual acceptable node it finds.

5 Related Work

Statistical resource guarantees: Previous work on application profiling [23] and resource overbooking have used intensive offline-analysis techniques for determining application placement at the single-node level to meet quality of service requirement levels. Our work targets online node-level behavior observations instead of offline application-level profiling. Recent work in Computational Markets [21] has placed importance on the prediction of consumer-oriented resource costs in a market-based pricing system to maintain quality of service levels. Such challenges are more on the socio-economic realm of distributed computing. Our work focuses more on the absolute capacities available on nodes in a large distributed system.

Resource discovery: SWORD [15] is a scalable resource discovery service deployed on PlanetLab that uses a DHT underlay to store node statistics. While the focus of SWORD is on providing scalability in terms of querying, we have also addressed the issue of scalability in data collection and propagation. While SWORD excels at finding sets of suitable nodes meeting instantaneous requirements for initial service placement, our approach is geared towards meeting statistical requirements.

Resource discovery in dynamic desktop grid environments has been explored in [10]. They use peer-to-peer query forwarding, which is poor for locating large groups of acceptable nodes. Our work instead emphasizes propagation of node behavioral information in a hierarchy to enable the quick discovery of large numbers of nodes.

The use of content-addressable networks (CANs) for resource discovery has been covered in [9]. Advertisements for jobs are inserted into the CAN in a decentralized fashion. Resource requirements are loosely defined, dynamism is not addressed, and single values for load are used.

Aggregation: Recent work on Information Planes [6] and Astrolable [19] provide frameworks for scalable deployments of information systems. This work has shown hierarchical aggregation to be a useful model for scalability, but in a more general context. Similarly, SDIMS [24] mentions aggregation as a key model for scalability, and specifies language constructs to support aggregation. While these systems provide a general aggregation framework, our focus is to solve the specific aggregation problem for resource usage distributions, and a key contribution of our work is to show the use of resource bundles for this purpose.

Historical data and prediction: Previous work on availability prediction [13] has studied patterns of availability and the application of various predictors. Availability prediction is a special case of prediction, with only two possible node states. Also, resource capacity prediction can be viewed as more of a reliability paradigm than availability. Histograms were used in workload prediction for multi-tier Internet applications in [22]. Such prediction is complimentary to our techniques for statistical guarantees. Statistical demand profiles similar to our requirements for applications are presented in [20].

6 Conclusions and Future Work

In this paper, we addressed the problem of scalable resource discovery in large loosely-coupled distributed systems. A key problem in these systems is that besides inter-node heterogeneity, they also show high degree of intra-node dynamism, where the resource capacities of individual nodes can vary drastically over time. This dynamism means that selecting nodes based only on their recently ob-

served resource capacities can lead to poor deployment decisions resulting in application failures or migration overheads. However, most existing resource discovery mechanisms rely only on recent observations to achieve scalability in large systems.

We proposed the notion of a resource bundle—a representative resource usage distribution for a group of nodes with similar resource usage patterns—that employs two complementary techniques to overcome the limitations of existing techniques: resource usage histograms to provide statistical guarantees for resource capacities, and clustering-based resource aggregation to achieve scalability. Using trace-driven simulations and data analysis of a month-long PlanetLab trace, we showed that resource bundles are able to provide high accuracy for statistical resource discovery, while achieving high scalability. We also showed that resource bundles are ideally suited for identifying group-level characteristics such as finding load hotspots and estimating total group capacity.

As part of future work, we intend to explore the use of resource bundles for multi-resource discovery. The problem of determining the optimal number of clusters for a live hierarchical aggregation algorithm is also an active part of our future work.

7 Acknowledgments

We would like to thank Arindam Banerjee for discussions and pointers to the code for the clustering algorithm, and Rohini Prinja for help with setting up the PeerSim simulator.

References

[1] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the 5th ACM/IEEE International Workshop on Grid Computing*, 2004.

[2] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11), 2002.

[3] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, S. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the grid using apples, 2003.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *Proceedings of ACM SIGCOMM*, Aug. 2003.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, July 2003.

[6] B. Chun, J. M. Hellerstein, R. Huebsch, P. Maniatis, and T. Roscoe. Design Considerations for Information Planes. In *First Workshop on Real, Large Distributed Systems (WORLDS '04)*, Dec. 2004.

[7] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems*, June 2003.

[8] S. Guha, N. Daswani, and R. Jain. An experimental study of the skype peer-to-peer voip system. In *IPTPS*, 2006.

[9] A. Gupta, D. Agrawal, and A. E. Abbadi. Distributed resource discovery in large scale computing systems. In *Proceedings of the The 2005 Symposium on Applications and the Internet (SAINT'05)*, pages 320–326, 2005.

[10] A. Iamnitchi and I. Foster. On Fully Decentralized Resource Discovery in Grid Environments. In *International Workshop on Grid Computing*, Denver, Colorado, Nov. 2001. IEEE.

[11] J.-S. Kim, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman. Using Content-Addressable Networks for Load Balancing in Desktop Grids. In *Proceedings of 16th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, June 2007.

[12] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. 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*, 2004.

[13] J. Mickens and B. Noble. Exploiting Availability Prediction in Distributed Systems. In *Proceedings of USENIX NSDI 2006*, May 2006.

[14] T. Nakajima, I. Satoh, and H. Aizu. A Virtual Overlay Network for Integrating Home Appliances. In *Proceedings of the 2002 Symposium on Applications and the Internet (SAINT'02)*, pages 246–253, 2002.

[15] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Distributed resource discovery on PlanetLab with SWORD. In *First Workshop on Real, Large Distributed Systems (WORLDS '04)*, Dec. 2004.

[16] D. Oppenheimer, B. Chun, D. Patterson, A. C. Snoeren, and A. Vahdat. Service Placement in a Shared Wide-Area Platform. In *Usenix Annual Technical Conference*, June 2006.

[17] K. Park and V. S. Pai. Comon: a mostly-scalable monitoring system for planetlab. *SIGOPS Oper. Syst. Rev.*, 40(1):65–74, 2006.

[18] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of IEEE International Symposium on High Performance Distributed Computing*, July 1998.

[19] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[20] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak. Statistical Service Assurances for Applications in Utility Grid Environments. Technical Report HPL-2002-155, HP Labs, 2002.

[21] T. Sandholm and K. Lai. A statistical approach to risk mitigation in computational markets. In *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pages 85–96, New York, NY, USA, 2007. ACM Press.

[22] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic Provisioning of Multi-tier Internet Applications. In *Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC 2005)*, June 2005.

[23] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI'02)*, December 2002.

[24] P. Yalagandula and M. Dahlin. A Scalable Distributed Information Management System. In *Proceedings of ACM SIGCOMM 2004*, 2004.

[25] S. Zhong. `http://www.cse.fau.edu/~zhong/software/index.htm`.

[26] S. Zhong and J. Ghosh. A comparative study of generative models for document clustering. In *SDM Workshop on Clustering High Dimensional Data and Its Applications*, 2003.