Resource Bundles: Using Aggregation for Statistical Large-Scale Resource Discovery and Management

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

This work was supported in part by an NSF CAREER Award CNS-0643505.

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 a 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 mainly 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, while achieving high scalability. We also show that resource bundles are ideally suited for identifying group-level characteristics (e.g. hot spots, total group capacity). To automatically parameterize the bundling algorithm, we present an adaptive algorithm that can detect online fluctuations in resource heterogeneity.

Index Terms

resource discovery, aggregation, resource management, machine learning

I. Introduction

Recent years have seen increasing use of loosely-coupled distributed platforms for scientific computation [1]–[3], data sharing and dissemination [4]–[6], and experimental testbeds [7]. Examples of such large-scale platforms include volunteer computation grids such as SETI@home (over 3.6 million participant machines), P2P systems such as Kazaa (over 30 million users) and Kad networks (over 2 million users). While such platforms are highly attractive due to their low deployment cost and inherent scalability, they are also highly heterogeneous and dynamic [8]. 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 [8]–[11] rely on the recently observed 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 [12] 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 [12], 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 [9], [10]). 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. A 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 for scalability. To handle the parameterization of the clustering algorithm, we present an adaptive algorithm to detect the amount of heterogeneity in the system and show its ability to adjust to fluctuations in an online fashion.

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. 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 demand 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 hot spots and estimating total group capacity (within 8% of actual values).

II. 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.

A. 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 an 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 [13].

B. 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 [12] 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 disruption 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.

C. 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 corresponding 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 and capture long-term vs. short-term trends.

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 how a statistical requirement can be mapped to a resource capacity histogram (with p% of the capacity observations to the right of the vertical line corresponding to capacity c). 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.

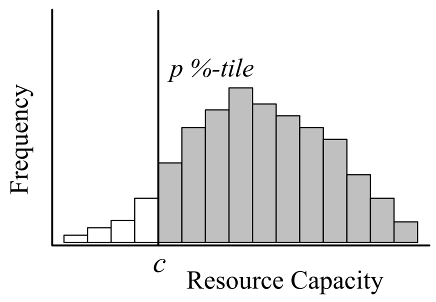


Fig. 1. Mapping a statistical requirement to a resource capacity distribution.

Using histograms to represent resource usage data has two primary advantages. First, requirement percentiles (corresponding to p) for a particular resource capacity are now straightforward computations from the given histograms. Second, histograms help us preserve all usage data, so that even if different applications specify different resource capacity requirements with different tolerances, these can be easily captured using the same histogram representation.

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 reduce storage overhead at each node, instead of maintaining the entire historical resource usage trace, one could choose to preserve only the histograms themselves (along with a decay function), thereby reducing the overhead by a factor proportional to the measurement frequency.

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.

III. 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, making it 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.

This raises the following questions. Can we use these representations in a scalable manner to make better resource discovery decisions 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?

A. Resource Aggregation

Aggregation [13], particularly hierarchical aggregation [14], 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.

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. 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. This 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 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, which shows the aggregation of two nodes, one skewed towards low capacity and the other skewed towards high capacity. We see that the average representation loses this information about the individual nodes and appears to represent a set of bimodal nodes. While averaging allows the estimation of overall capacity of the group of nodes, it is a poor representation of individual node-level behavior. This is because 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 a highly inaccurate view of individual node resource capacities.

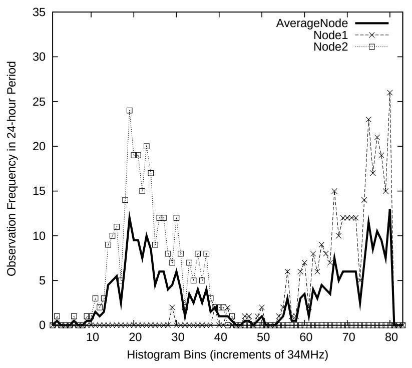


Fig. 2. Naive approach to aggregation by averaging resource capacity distributions.

B. *Defining Resource Bundles*

To account for the heterogeneity of nodes, we define the notion of a *resource bundle:* an aggregation of a group of nodes with *similar* resource capacity distributions. By combining only similar nodes together, such an aggregation process will preserve the individual node distributions more accurately. 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. The question is how can we identify such groups of similar nodes to construct a resource bundle, and how can we compute a representative to accurately represent the nodes in the bundle?

1) Aggregation via Clustering: 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 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.)

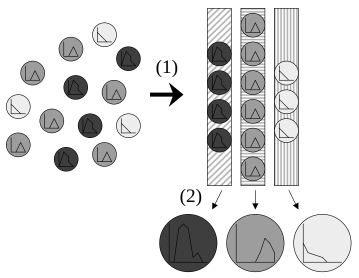


Fig. 3. Constructing Resource Bundles.

A clustering algorithm that meets the above requirements is the *multinomial model-based expectation maximization (EM)* clustering algorithm [15]. 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. Then, the set of all documents is represented as a mixture of multinomial 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 [15]: $P(d_i|\lambda_j) = \prod_l P_j(w_l)^{n_{il}}$, 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 multinomial distribution, it is able to characterize the common statistical properties of the nodes in a cluster in a compact manner.

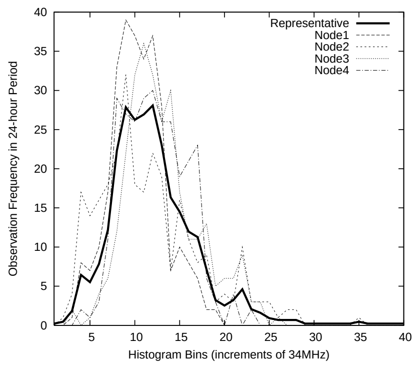


Fig. 4. Four nodes bundled by CPU capacity, producing a bundle representative

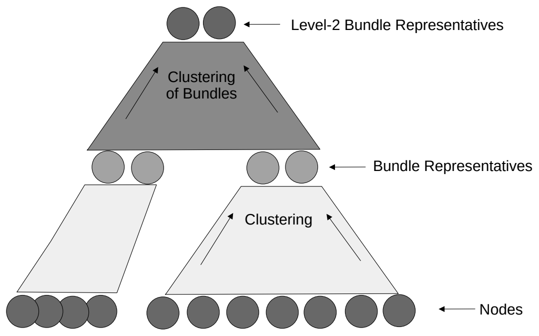


Fig. 5. Hierarchical aggregation

2) Bundle Representatives: As described above, the multinomial model-based EM algorithm associates each resource bundle with a multinomial distribution that captures the statistical properties of the nodes that are members of the bundle. This multinomial distribution can be thought of as the *bundle representative:* an aggregate distribution that is a compact representation of the individual node 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; 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. The representative closely matches the individual node distributions. In Section IV, we will quantify the closeness of the representative to its member distributions, and its effectiveness in resource discovery, hot spot detection and capacity estimation.

C. 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 particularly 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 propose the use of *hierarchical aggregation* 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 representative clustering process.

Note that this recursive aggregation must incorporate individual bundle cardinalities during the clustering process to minimize the loss of representative data. 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. 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 bundle histogram H_i having cardinality w_i, we present it to the clustering algorithm as $W_i = H_i w_i$.

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. Our ability to predict group-level capacities results in another high-level ability to detect hot spots within large distributed systems.

D. Implementation Considerations

In this section, we describe the implementation details of hierarchical resource bundling over a set of distributed nodes: how data would be exchanged, aggregated, and propagated. We assume an implementation on top of a hierarchical overlay as described in our system model in Sec II-A.

First, assume only one resource type being measured for simplicity (e.g. CPU). Different resource types are assumed to be handled independently of each other. As an aside, note that Resource Bundles can be used for multi-dimensional resource discovery, provided that the individual resources are bundled independently of one another. One could then take the intersection of multiple different resource requirements and use that as a result. However, this assumes statistical independence between the different resource types. We leave the general problem of multi-dimensional resource discovery as future work.

As nodes compile their own resource usage distribution histograms, they send all histograms up the hierarchy, including those received from their children. If this process would continue unrestricted, it would be a fully-informed algorithm, with the root receiving all node resource usage histograms. However, to achieve scalability, we restrict the amount of propagated data by introducing aggregation at different points in the hierarchy as follows. When the number of histograms reaches some threshold B_t, then the resource usage histograms are aggregated by the resource bundles algorithm, producing B_a aggregate bundles (representative histograms) as output. Then the B_a bundles are sent upward instead of (greater than) B_t histograms, thus applying aggregation for scalability.

The choice of B_t determines the maximum amount of resource bundle representatives (histograms) that can be sent upwards by any node in the system, and also determines how often the bundles must be aggregated. If this number is too small, then the clustering algorithm would be run too frequently and on too few histograms, resulting in an underfitting of the data for the formation of bundle representatives. If this number is too high, then the histograms would be aggregated infrequently (as they propagated to the root), leading to unnecessary data transfers of large numbers of histograms. The rule of thumb we used for B_t was to define it as a small multiple of the number of clusters n_c in the system. For our simulations in Section IV-G, we chose that multiple to be 7; so that for 10 clusters, $B_t = 70$. The number of aggregate bundles produced B_a is identical to the number of clusters n_c. We present an adaptive algorithm to select this parameter n_c in section III-F.

Notice that in a hierarchical system, the bundling algorithm may receive as input all of the following: single-node histograms, first-level bundle representatives (e.g. representing tens of nodes) and high-level bundle representatives (e.g. representing thousands of nodes). Therefore we also maintained *bundle representative cardinalities* to appropriately weigh the formation of higher-level bundle representatives from these various levels of input data.

E. Analysis of Network Overhead

We now analyze the network overhead complexity of our Resource Bundles algorithm in terms of the number of messages sent in a complete propagation on a hierarchical overlay, as described above. We compare this overhead complexity to that of a baseline algorithm PropagateAll (described in more detail in Section IV) that simply propagates everything to the root in the hierarchy.

Intuitively, with an extreme parameterization of Resource Bundles (a small value of B_t), aggregation would take place at every (internal) level in the tree, so that a constant number of histograms (B_a) is propagated upward by each node. Therefore, this is an $O(n)$ algorithm in terms of the number of messages being propagated in the system (with n being the number of nodes in the system). In a real implementation of Resource Bundles, aggregation need not take place at every level of the tree. The aggregation points would depend on the degree of each node and at the locations in the tree where the thresholds (B_t) are exceeded. Deriving closed-form equations for the complexity of Resource Bundles in the general case proves to be difficult, as the degree of the tree nodes and the parameters of the aggregation together may create an oscillatory effect on the frequency of the aggregation as we traverse a tree. So to show Resource Bundles is an $O(n)$ algorithm, we introduce a simplified experiment where we have a balanced-tree assumption with respect to the hierarchy construction[1].

[1]We also show this result using simulations of realistic systems with diverse topologies in Section IV-G.

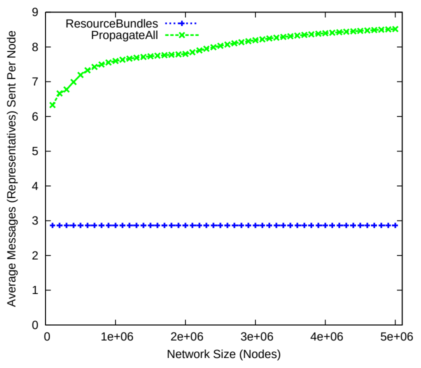


Fig. 6. Resource Bundles shows $O(n)$ complexity

In Figure 6, we show the average number of messages sent per node in the system with the balanced-tree assumption, for system sizes between 100,000 and 5 million nodes. This number for Resource Bundles remains constant at 2.87 for all such system sizes.

As a comparison, PropagateAll by definition is $O(n\ log(n))$. The histograms at the bottom of the tree must traverse the height of the tree up to the root, which is a path of length $log_d(n)$, where d is the average degree of a node.

Besides the overall bandwidth in the network, bandwidth consumption at the upper levels of the tree can be a concern, as too many messages near the root may present a bottleneck to tree-based algorithms. Under Resource Bundles, any node will receive $(B_t - 1)d$ messages in the worst case, or 414 messages as configured in Figure 6. In the same tree configuration at 5 million nodes, PropagateAll nodes directly below the root received on average 833,332 messages per propagation, which is over 2,012 times the worst-case bandwidth of Resource Bundles, and in this system was over 3,800 times the actual bandwidth of Resource Bundles, which averaged 218 messages received at the same level.

F. Adaptive Algorithm for Parameter Selection

One must consider the importance of choosing an appropriate value for the number of clusters (n_c) in the resource bundling algorithm. If too small a value is chosen, the bundle representatives will poorly represent their constituent nodes due to high heterogeneity. If too large a value is chosen, then these techniques for aggregation will not be effective due to the large overhead in transferring nearly as many bundle representatives as there are nodes themselves.[2] Therefore we must carefully employ techniques to have the system determine this parameter robustly without the need for administrative intervention.

[2]We show this tradeoff empirically in Section IV-E.

To automatically select the appropriate number of clusters value in a live system, we propose an adaptive hill-climbing algorithm for the selection of n_c. The algorithm works as follows. The resource discovery query system introspectively monitors the quality of its decisions (i.e., the choice of nodes during resource discovery) over time. It returns its decisions based on the current value of n_c, but also stores decisions that would have been made for other, nearby values of n_c. It then evaluates this set of possible decisions and tries to maximize an objective function in order to achieve better results in the future by choosing a progressively better value for n_c.

Our objective function to determine the goodness of the result as compared to other values of n_c is defined as $\beta(p, r, n_c) = F1Score(p, r) - \delta n_c$, where $F1Score(p, r) = \frac{2pr}{p+r}$, where p is the precision of resource discovery (i.e. the fraction of nodes satisfying their requirement from the selected set of nodes), r is the recall of resource discovery (i.e. the fraction of nodes discovered in the system out of the total nodes in the system that would have satisfied the requirement), and δ is the threshold of the minimum desired improvement in the $F1Score$ per unit increase in the number of clusters n_c, i.e., $\delta = minDesired(\frac{\triangle F1Score}{\triangle n_c})$. The $F1Score$ is the ***harmonic mean*** of the precision and recall, and is commonly used in machine learning techniques as an objective function to determine the high-level accuracy of clustering decisions.

For instance, a decision to allocate 30 nodes for a period of 24 hours under $n_c = 20$ could be evaluated 24 hours later to determine the actual precision and recall. The algorithm also evaluates the alternative decisions that could have been made given other surrounding values of n_c such as $n_c = \{17, 18, 19, 21, 22, 23\}$. In retrospect, if the objective function would be maximized for $n_c = 17$, then the value of n_c would be changed to 17 for the next bundling cycle, and the algorithm will do another introspective evaluation some time later.

In our analysis, we chose $\delta = 0.005$. The choice of δ, unlike the choice of n_c, is not system-dependent. The inclusion of the $-\delta n_c$ term in β avoids local maxima values and allows the algorithm to more accurately determine the true knee of the objective function curve. For instance, a selection of $n_c = 45$ might not seem to be a bad selection, but if the same $F1Score$ was available for $11 \leq n_c \leq 45$, then 11 is a much better choice for aggregation overhead reasons.

G. Alternate Compression Techniques

One of the benefits of resource bundles is their ability to compress node-level resource usage representation data. There exist other techniques for compressing this data that we consider below as alternatives to resource bundles. We will discuss their tradeoffs and explain our choice of resource bundles in our context.

A simple technique for compression is general lossless compression over the histograms, e.g., LZMA. Running such a lossless compression algorithm over usage histograms, while reducing the overall bandwidth requirement, will *cause the amount of data to grow in proportion to the number of histograms* at higher-level nodes in the tree, placing more load on them. Even with PropagateAll using compression under the scenario described in Section III-E, its bandwidth at the root is over 48 Mbps, assuming one propagation per minute. On the other hand, resource bundles can be combined recursively many times over, thus *keeping the amount of data bounded* for nodes at all levels of a propagation tree. The worst-case bandwidth for any node in Resource Bundles under the same scenario is 11.64 Kbps uncompressed. Further note that we can apply a lossless compression algorithm to resource bundles as well, further reducing their bandwidth requirement (4.15 Kbps for the above scenario). Finally, we will demonstrate in Section IV that even with recursive aggregation, resource bundles are highly accurate with minimal data loss.

Another approach for compactly approximating a histogram would be to use a specific distribution (e.g., Chebyshev or Gaussian) to represent each node's usage data. However, a specific distribution would not be able to accurately capture the (arbitrary) shape of the whole histogram in general. As a result, it might be accurate for specific percentile values, but would result in less accuracy for an arbitrary percentile value, thus reducing the flexibility of resource discovery. It would also render our group-level capacity estimations inaccurate.

It is important to note that we are bundling *resource usage representations (histograms) only*. Other host-specific information (such as host IP address, geographic location, etc.) is largely static or changes very infrequently. Such data need not be updated and propagated along with dynamically changing resource usage data. Rather, such static data could be accessed in a much more efficient manner without the need for frequent updates. For example, it could be stored in a distributed storage or file system, and referred using small IDs or keys. These IDs could then be passed around in the Resource Bundles implementation, and lookups done on demand. To avoid a scalability issue with passing large numbers of IDs around the system, nodes in the hierarchy could easily cache sets of IDs (i.e., bundle memberships) and updates would only consist of changes in these memberships, along with their bundle representatives.

IV. Evaluation

We now evaluate the performance of using resource bundles for resource discovery, capacity estimation and hot spot 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 IV-G.

A. Data Analysis Methodology

We used a PlanetLab trace obtained by CoMon [16] from February 2007 for our experiments. CoMon runs a daemon on each PlanetLab node and 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[3]. We used a time period of 24 hours as the desired time window[4], thus computing node histograms on a 24-hour period, one per day. Each histogram consisted of 100 bins[5] (with an overflow bin), each one representing about 34 MHz, with 3.4 GHz assumed as the maximum CPU capacity in PlanetLab.

We used MatLab as our main tool for data analysis incorporating an already implemented Matlab package for the multinomial model EM clustering algorithm [18]. 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. In a simple benchmark used to test the computational overhead of the clustering algorithm in MatLab, we determined the average running time to be 0.72 seconds, indicating that the algorithm has very *low computational overhead.*

[3]Churn is a widely-studied phenomenon in wide-area distributed systems and is beyond the scope of our paper, but is being considered as future work.

[4]We use a 24-hour time period for clarity of exposition, but maintaining histograms for other time periods is a straightforward exercise.

[5]One might consider variable-sized bins for compression reasons [17], but the resource bundles algorithm requires fixed boundaries for the histogram bins across the entire system.

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.

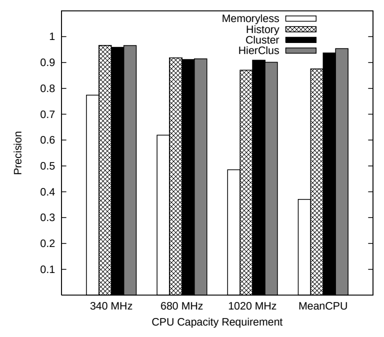
For the purposes of our experiments, we defined the statistical requirement r={CPU, c, p, 24hrs}, with different values of c and p. The goal of the algorithms was to find all nodes meeting requirement *r* on the 427-node PlanetLab trace. Notice we did not specify how many nodes an application needs for its deployment. Instead of using this as a parameter in our 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. Note that the propagation frequency in an implementation would be higher than once per day (e.g., every 5 minutes) to avoid staleness. Here we use one histogram per day to ensure our time windows do not overlap with previously used histograms for evaluation.

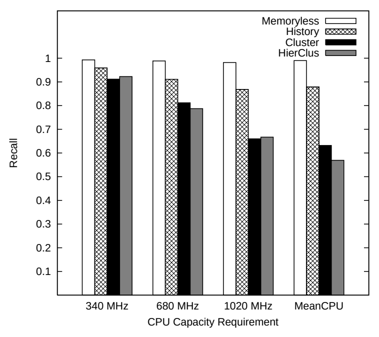
2) Comparison Algorithms: We compared the following resource discovery algorithms:

• ***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.

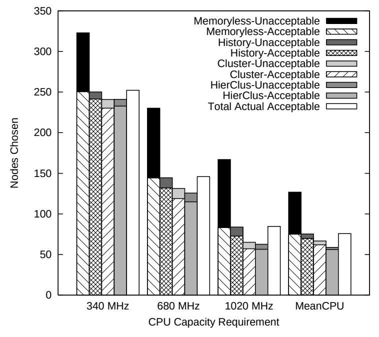
• ***History***: This is a centralized algorithm with global historical knowledge of the entire system. It maintains complete 24-hour CPU capacity histograms for each node. Each node histogram is individually examined to determine which nodes meet the desired statistical requirement r, thereby determining 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.



(a) Precision



(b) Recall



(c) Nodes Discovered

Fig. 7. Aggregation compared against baseline resource discovery algorithms (95^{th} percentile).

• ***Aggregation (Cluster)***: 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. For each bundle, if its representative meets the desired statistical requirement r, all of its members are selected as acceptable nodes.

• ***Hierarchical Aggregation (HierClus)***: This algorithm uses recursive bundling over a 2-level hierarchy consisting of 420 PlanetLab nodes divided into 6 random groups of size 70. The hierarchical aggregation algorithm first reduces each group of nodes to k first-level resource bundles. All the first-level resource bundles from across all groups are then further aggregated into k second-level resource bundles. For resource discovery purposes, the algorithm examines these second-level bundle representatives and if a bundle representative H_i meets requirement r, then all nodes represented by H_i (recursively) are selected as acceptable.

3) Evaluation Metrics: The accuracy of resource discovery was measured using two metrics: $Precision = \frac{n_{acc}}{n_{tot}}$ and $Recall = \frac{n_{acc}}{N_{acc}}$, where, 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 it can locate acceptable nodes in the system.

B. Accuracy in Resource Discovery

Figure 7 compares the accuracy of our single-level and hierarchical aggregation algorithms using 10 clusters[6] 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 over all possible requirements (for each of the 100 histogram bins).

As seen from Figure 7(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 historical information to make statistically accurate decisions. The algorithms using 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 7(b), which shows Memoryless had the highest recall, while History came in second, and Cluster consistently had the worst recall among the three. 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(c), 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

[6]We investigate the impact of number of clusters on aggregation accuracy in Section IV-E.

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.

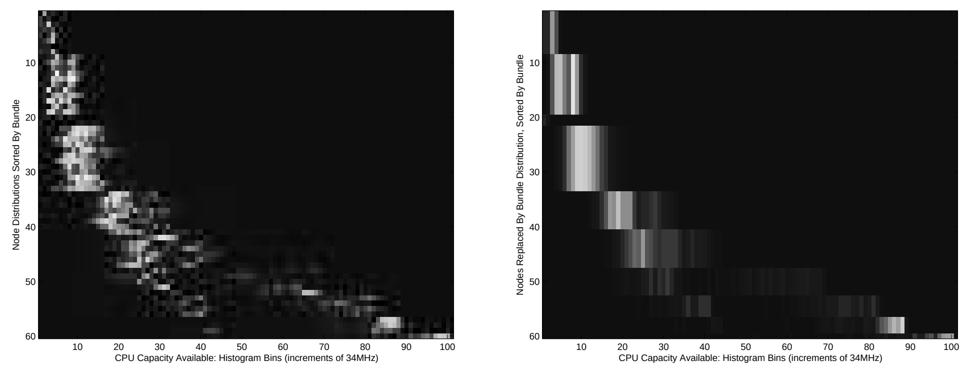
Figure 7 also compares HierClus with Cluster for precision and recall statistics. Precision does not suffer from the recursive aggregation in HierClus, while the recall is slightly lower (about 6% for MeanCPU). This shows that the loss of accuracy in hierarchical clustering is small compared to a single-level clustering.

In summary, our precision and recall evaluations show that resource discovery using *our resource bundle approach is able to find a choice of nodes similar in quality to those discovered by a fully-informed history-based algorithm*, although its set of chosen nodes is smaller, thereby missing a few potentially acceptable nodes.

C. Identifying Group 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.

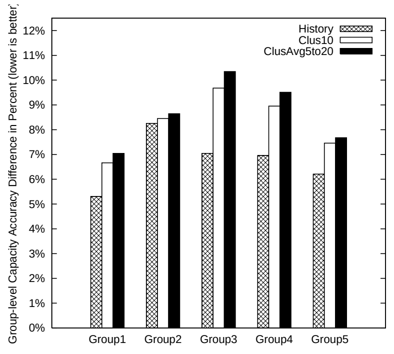
To establish proximity-based groupings of nodes for our experiment, we hand-selected 5 geographically distributed group leaders in our PlanetLab trace. These leaders were respectively from UMASS Amherst, UFL, UT Austin, UWash, and UMN, representing different US regions. Pairwise pings between these leaders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



(a) Nodes: sorted by bundle rep (b) Nodes: replaced by bundle rep

Fig. 8. Bundle representatives can be used to detect hot spots in the network.

total of 300 responsive PlanetLab nodes. We formed 5 groups of 60 nodes each based on their proximity from the selected group leaders.



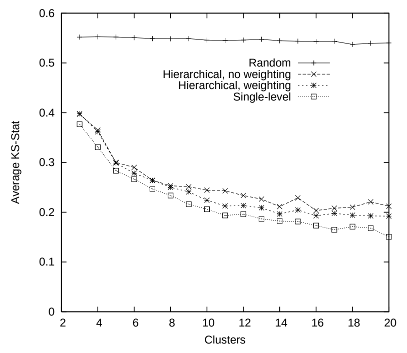


Fig. 9. Accuracy of capacity estimation.

Fig. 10. Information loss for aggregation.

1) Load Hot Spot 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 bundle representatives we obtain by our clustering algorithm. As seen in the 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 (nodes 9-19) appears to have very low spare CPU capacity, and could potentially be overloaded. Using such group-level views can allow administrators to identify potential load hot spots by using appropriate thresholds.

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 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, group capacities are estimated by taking a weighted sum of the mean capacities of the bundle representatives; for the history-based estimation algorithm, the mean capacities of individual nodes are added.

Figure 9 shows our results. 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).

D. 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.

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; the more similar the two distributions are, the closer the statistic is to 0.

For our evaluation, we compare the K-S statistic for the following algorithms: single-level aggregation, hierarchical 2-level aggregation, hierarchical 2-level unweighted aggregation (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 10 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, this result still shows that the loss of information is relatively low considering that 420 distributions have been reduced to 10 distributions. The results from previous sections 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. Also, 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.

E. *Effect of Clustering Parameters*

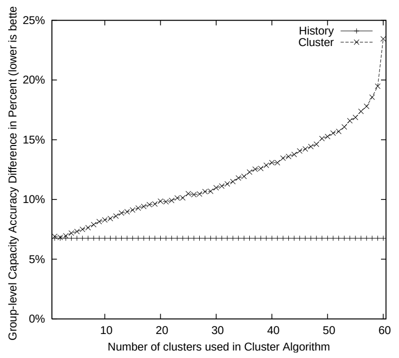


Fig. 11. Fewer clusters are better for capacity estimation.

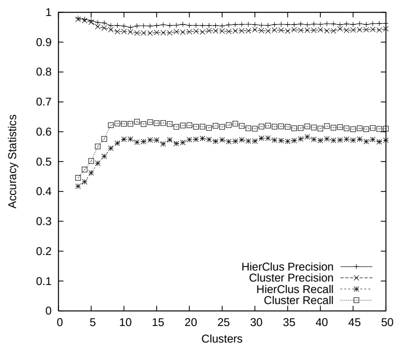


Fig. 12. 10 clusters is appropriate. (MeanCPU, $p = 95\%$)

We next examine the impact of the number of clusters used by the clustering algorithm to generate the desired set of resource bundles. Figure 11 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 10. However, this inaccuracy occurs because the 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[7]. With a high number of clusters, this difference gets amplified due to the additive nature of capacity estimation.

Intuitively, since the resulting representation of a single resource bundle from the multinomial EM algorithm is actually a prediction model, each resource bundle has a certain amount of error, or noise. Using more bundles is akin to adding more noise into the sum for group capacity measurements, leading to more inaccuracies with higher numbers of bundles used.

Together, the results from Figures 10 and 11 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 12 that shows diminishing returns in the accuracy of resource discovery over the number of clusters (n_c) beyond the knee of the curve (at $n_c = 10$). In general, this value is dependent on the dynamism and heterogeneity of the system.

F. Adaptive Algorithm for Parameter Selection

We applied our adaptive algorithm described in Section III-F to the PlanetLab trace. First, we applied our β objective function to the overall month-long trace to observe its derivation of the ideal number of clusters (n_c) in the system from the peak in the curve. Second, we applied β to a live rendition of the PlanetLab trace, and also subjected it to a systematic fluctuation in heterogeneity in order to observe its ability to detect this change.

Figure 13 plots β against precision and recall from figure 12. This algorithm therefore approaches the appropriate value for the “knee” of the curve that provides the best benefit to the system in terms of resource discovery accuracy and overhead.

Figure 14 shows our function β working adaptively in the Feb 2007 PlanetLab trace. After each day, the adaptive algorithm considered $n_{c,NEW} : n_c \pm 5$ based on query results evaluated

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

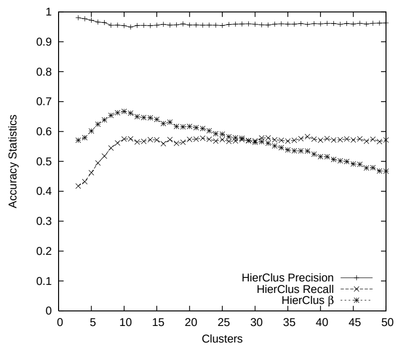


Fig. 13. Heuristic function β used to determine the number of clusters parameter n_c. β is a function of the $F1Score$ (from Precision p and Recall r, as shown), and δ, the threshold minimum desired increase in $F1Score$ per unit increase in n_c.

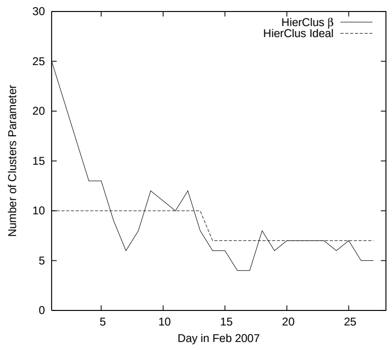


Fig. 14. Determining n_c using adaptive function β. The n_c parameter is incorrectly initialized at 25 clusters, quickly converging near the ideal. At $t = 14$, some heterogeneity is removed from the system to bring the ideal n_c down to 7. The adaptive algorithm converges to the new ideal value.

from the previous day. A new value for n_c was chosen at the end of each day. We initialized $n_c = 25$, more than double the ideal value of 10 (from Figure 13). However, as seen from the figure, our algorithm was able to converge to the ideal value of 10. We also injected a fluctuation in the amount of heterogeneity in the system at $t = 14$ by removing 3 of the 10 observed bundles of nodes in the system and replacing them with a distribution from another existing node, effectively reducing the heterogeneity in the system but keeping the number of nodes constant. This fluctuation reduces the ideal value of n_c to 7, and once again, our algorithm is able to find this optimal value.

The results show that our adaptive algorithm (β) successfully converges to the ideal value for n_c. Please note that the observed fluctuations around the ideal n_c are due to daily fluctuations that do not appear in Figure 13 (since Figure 13 is derived from all 28 days of the trace combined).

G. Trace-Driven 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 realistic topology. We have based our simulator on PeerSim [19], a widely used simulator for distributed protocol evaluation. We have used our month-long PlanetLab trace to drive these simulations. A hierarchical overlay was implemented using *WireInetTopology* in PeerSim. The depth of the hierarchy varied from 5 to 7 levels. We generated 10 random topologies for each network size and took an average over them. Single nodes may now be bundled together with multi-level bundles, so we define a *bundle* as representing 1 or more nodes.

At each simulator cycle, each node propagates its bundle list upwards to its parent. 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 ran our resource bundle-based aggregation algorithm over the simulated topologies. We compared it to a baseline ***PropagateAll*** algorithm, which uses a threshold $B_t = \infty$, so that no aggregation takes place and 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.

1) Data Transfer Overhead: We measured the data transfer that occurred during a complete system-wide propagation to the root node in the hierarchy. We generated 10 random topologies for each network size. Data is measured in the number of bundles (directly proportional to bandwidth) sent over the network. Figure 15 measures overhead by the size of the network. The overhead of Aggregation ($B_t = 70, B_a = 10$) increases at a slower pace than PropagateAll, with only about 48-61% messages required across the different network sizes.

2) Query Latency: We now analyze query latency from our trace of 11,529 PlanetLab node distributions (427 nodes$*$27 days as in Section IV-A.1). We chose not to fix an application-desired number of acceptable nodes to be found; instead, we measured how many nodes were found given a time-to-live value measured in hops. We used the requirement $\{$CPU, 680MHz, 95, 24hrs$\}$ to determine the actual acceptable nodes. Our results are the average of queries injected at 500 random points in the network. Queries are answered on the node on which the query was injected and can be propagated further to other nodes. Bundles were chosen based on their histograms and the number of acceptable nodes found were evaluated as in previous sections.

Figure 16 shows that Aggregation finds nearly as many nodes as PropagateAll (within 11%), but with about half the data transfer overhead in the same hierarchical structure. PropagateNone, which has no data overhead, performs much more poorly, returning only about 1-15% of nodes

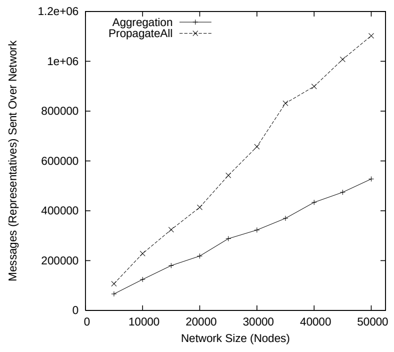


Fig. 15. Data transfer overhead.

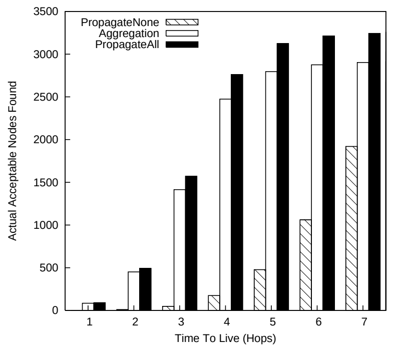


Fig. 16. Nodes found by query latency.

up to 5 hops. It also has high query-time overhead; it must query each individual node.

Overall, these results show that *using aggregation provides a good tradeoff between data dissemination overhead and query-time overhead, with fairly high accuracy.*

V. Related Work

Statistical resource guarantees: Offline profiling [20] has been used for application placement to meet statistical QoS levels. Our work targets online node-level as well as group-level capacity observations for statistical resource discovery. Computational Markets [21] has focused on the prediction of consumer-oriented resource costs in a market-based system to maintain QoS levels. We focus more on resource discovery than socio-economic dynamics.

Resource discovery: Condor [11] employs centralized matchmaking; our focus is on decentralized discovery. SWORD [9] is a resource discovery service deployed on PlanetLab that uses a DHT underlay to store load metrics. While SWORD provides scalability in querying, we have also addressed scalability in data collection and propagation. Several recent approaches have explored resource discovery in dynamic desktop Grid environments [8], [10], [22]. Most of these approaches use peer-to-peer query forwarding, and focus on finding single nodes for individual application tasks. Our work instead emphasizes hierarchical propagation of node resource usage information to enable the quick discovery of large groups of nodes. Also, while existing resource discovery work focuses on finding suitable nodes meeting instantaneous requirements, our approach is geared towards statistical requirements. Resource 'bags' [23] have been used in static and inter-node contexts; dynamic intra-node metrics is our focus.

Aggregation: Information Planes [14], Astrolabe [24], SDIMS [13] and San Fermin [25] provide frameworks for scalable deployments of information systems and show hierarchical aggregation to be a useful model for scalability. While these systems provide general aggregation frameworks, they do not provide specific aggregation functions. Our focus is on solving the specific aggregation problem for resource usage distributions.

Historical data and prediction: Prediction has been used in several contexts such as availability prediction [26], Web workload prediction [27], and workload prediction for multi-tier Internet applications [28]. Network Weather Service [29], [30] uses tournament predictors to accurately predict trends in resource usage levels. However, their predictions are limited to the next measurement point, while our techniques are flexible in the temporal and statistical descriptions of resource usage. Such prediction is complimentary to our use of resource usage profiles and we can incorporate prediction for greater accuracy in providing statistical guarantees.

VI. Conclusion

In this paper, we addressed the problem of scalable resource discovery in large-scale systems. The presence of node-level dynamism means that selecting nodes based only on recently observed capacities can lead to poor deployments resulting in application failures or migrations. However, existing resource discovery techniques rely only on recent observations to achieve scalability.

We proposed the notion of a resource bundle 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. We presented an adaptive algorithm that detects fluctuations in heterogeneity in order to parameterize the clustering-based resource bundles algorithm. Using trace-driven simulations and data analysis of a 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 hot spots and estimating total group capacity.

References

[1] D. Anderson, “BOINC: A System for Public-Resource Computing and Storage,” in *GRID 2004*.

[2] 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.

[3] I. Foster and C. Kesselman, Eds., *Grid2: Blueprint for a New Computing Infrastructure*. M. Kauffman, CA, USA, 2004.

[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. Cohen, "Incentives build robustness in BitTorrent," in *the 1st Workshop on the Economics of P2P Systems*, Jun. 2003.

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

[7] 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*, vol. 33, no. 3, pp. 3–12, Jul. 2003.

[8] A. Iamnitchi and I. Foster, "On Fully Decentralized Resource Discovery in Grid Environments," in *GRID 2001*.

[9] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Distributed resource discovery on PlanetLab with SWORD," in *WORLDS'04*, Dec. 2004.

[10] J.-S. Kim, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman, "Using Content-Addressable Networks for Load Balancing in Desktop Grids," in *HPDC'07*, Jun. 2007.

[11] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," in *HPDC'98*, Jul. 1998.

[12] 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*, Jun. 2006.

[13] P. Yalagandula and M. Dahlin, "A Scalable Distributed Information Management System," in *SIGCOMM'04*, 2004.

[14] B. Chun, J. M. Hellerstein, R. Huebsch, P. Maniatis, and T. Roscoe, "Design Considerations for Information Planes," in *WORLDS'04*, Dec. 2004.

[15] 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.

[16] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab," *SIGOPS OS Rev v.40 no.1*, 2006.

[17] J. M. Schopf, "A practical methodology for defining histograms for predictions and scheduling," in *NU Tech Rep.*, 1999.

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

[19] "PeerSim," http://peersim.sourceforge.net.

[20] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms," in *OSDI'02*, December 2002.

[21] T. Sandholm and K. Lai, "A statistical approach to risk mitigation in computational markets," in *HPDC'07*, 2007.

[22] A. Gupta, D. Agrawal, and A. E. Abbadi, "Distributed resource discovery in large scale computing systems," in *SAINT'05*.

[23] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, and A. A. Chien, "Efficient resource description and high quality selection for virtual grids," in *CCGRID'05*, pp. 598–606.

[24] 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.*, vol. 21, no. 2, pp. 164–206, 2003.

[25] J. Cappos and J. H. Hartman, "San fermín: aggregating large data sets using a binomial swap forest," in *NSDI'08*.

[26] J. Mickens and B. Noble, "Exploiting Availability Prediction in Distributed Systems," in *NSDI'06*, May 2006.

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

[28] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Dynamic Provisioning of Multi-tier Internet Applications," in *ICAC'05*.

[29] R. Wolski, "Experiences with predicting resource performance on-line in comp. grid settings," in *SIGMETRICS'03*.

[30] "Network Weather Service," http://nws.cs.ucsb.edu/ewiki/.