PRISM: Autonomous and Robust Resource Information Management for Shared Resource Platforms

Inseok Hwang, Hyukjae Jang, Hyunju Jin, Sungwon P. Choe,
Yongjoon Son, Jaesun Han, and Junehwa Song

CS/TR-2006-254
April 19, 2006

KAIST
Department of Computer Science

PRISM: Autonomous and Robust Resource Information Management for Shared Resource Platforms

Inseok Hwang, Hyukjae Jang, Hyunju Jin, Sungwon P. Choe,
Yongjoon Son, Jaesun Han, and Junehwa Song

Korea Advanced Institute of Science and Technology (KAIST)
{*inseok, hjjang, hyunju, sungwon, yjson, jshan, junesong*}*@nclab.kaist.ac.kr*

Abstract

Shared resource platforms like the Grid and PlanetLab are constructed with federated and distributed resources provided by different organizations. This motivates the need for decentralized resource information management that can tolerate unpredictable dynamics. In this work, we present PRISM, an autonomous and highly robust system for resource information management. The key design priciple of PRISM is to preserve the autonomous features of shared resource platforms in order to achieve robustness. Each node manages global resource information in an autonomous and fully decentralized way; performing updates and lookups with only local decisions. This enables PRISM to be highly robust against node failures or network partitions. PRISM is realized through our two major contributions: 1) the probabilistic dissemination protocol (PDP), which supports autonomous information dissemination guided by our unique logical hierarchy, and 2) vector-preserving aggregation (VPA), which achieves scalable aggregation and efficient searching of a huge volume of globally distributed resource information. We demonstrate the performance of PRISM through a suite of experimental results.

1. Introduction

Shared resource platforms such as the Grid [10], XenoServer [11], and PlanetLab [15] are being developed to provide a general environment for large scale distributed services such as P2P file sharing [17], CDN [18], and distributed scientific computing [20] to be deployed and evaluated in a cost-effective and time-efficient manner. Resource pools of these platforms are usually constructed with federated resources contributed by different organizations, so that resources are distributed not only geographically but also administratively. In other words, each organization usually retains ownership and management of its own resources.

Due to these distributed authorities over individual resources, centralized resource information management is politically difficult to apply to shared resource platforms. Moreover, their distributed authorities cause stability problems with their resource pools; for instance, each organization can reboot its machines, upgrade software, or change configurations autonomously, which

may lead to unexpected leaves of some resources from the resource pool. Also, since resources are geographically distributed, the resource pool is vulnerable to network partitions and failures which result in disturbances or unavailability of resources. These uncertainties are inevitable in shared resource platforms. Recently proposed resource information management systems [5] [8] for shared resource platforms take decentralized approaches by employing distributed hash tables [13] [22]. Their key design principle is that the nodes managing a subset of resource information can be different from the nodes managing others; each participating node is assigned a subset of the overall resource information to be responsible for. This approach may still suffer from group failures or network partitions, since they may lead to system-wide unavailability of the resource information stored at the troubled nodes. Even without failures, some high-demand resources may cause hot-spot problems, which again threaten the availability of the resource information.

In this paper, we propose PRISM, a highly robust, fully decentralized, and autonomous system for resource information management in shared resource platforms. The key philosophy of PRISM is that individual resource information is propagated to the whole system in probabilistic way. In the aforementioned environments of shared resource platforms, PRISM is designed to successfully find a requested resource node regardless of any failures or degradations (with the sole exception of when the resource itself doesn't exist at all). Two essential features of PRISM follow accordingly. First, all participating resource nodes are functionally identical and operate autonomously without other nodes' supervision. Second, each node maintains system-wide resource information as well as its own in scalable fashion. These two features enable a number of novel advantages in PRISM. For example, any node in PRISM can perform resource lookup just with its own local search. Moreover, any unexpected node leaves or failures cause hardly any information loss or functional degradation in the PRISM system. Essentially, the only information loss in the system caused by a failure is the precise resource status of the failed node itself, which is already no longer useful. These key advantages of PRISM are realized through our two major contributions: 1) *the probabilistic dissemination protocol* (PDP), which enables autonomous information dissemination guided by our unique logical hierarchy, and 2) *vector-preserving aggregation* (VPA), which achieves scalable aggregation and efficient searching of a huge volume of distributed resource information.

PDP is a probability-based protocol in which each node disseminates the updates of its local resource information to other nodes. However, probability-based dissemination may cause redundancies or starvation. Our solution is a *logical hierarchy*, the key uniqueness of PDP, which defines multi-level memberships of participating nodes with a tree structure. Note that since there are no actual representative nodes at each level of the hierarchy, the logical hierarchy is not vulnerable to node failures. The logical hierarchy guides the destinations and the frequencies of dissemination according to the nodes' memberships and a set of probability parameters which are

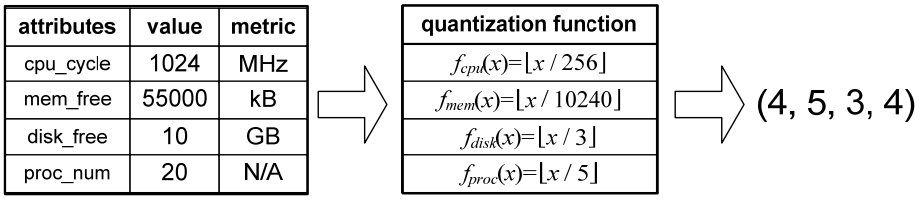


Figure 1: Example of node attributes, quantization functions, and the vector.

configured for each level of hierarchy. With the help of the logical hierarchy, PDP can propagate updates to the entire PRISM system in an efficient and scalable fashion.

VPA is an efficient aggregation mechanism for resource information by which every node can store and manage a large volume of global resource information collected by PDP. The key feature of our aggregation is that VPA aggregates nodes' information into a compact form, preserving the cross-attribute relations of individual resource nodes in order to support multi-attribute queries. This is made possible by VPA's unique method of resource description using *classes*, *vectors*, and *matrices*. The value of each resource attribute is quantized to one of a finite number of classes, and the vector is a set of classes of multiple attributes; a node's resource information with N attributes can be interpreted as a simple N-dimensional vector. Similarly, resource information of multiple nodes is represented as an N-dimensional matrix where each dimension is mapped to each attribute, so that an N-dimensional vector uniquely locates an entry of the matrix. In other words, the entry represents the count of nodes whose resource classes are identical, and essentially, the aggregation is simplified to additions and subtractions to or from entries in the matrix. Note that the matrix size is independent from the number of nodes aggregated. Therefore, VPA achieves highly scalable aggregation with a large number of nodes while still preserving cross-attribute relations, so that even multi-attribute queries can be resolved in a local node.

We proved the effectiveness of PRISM through a suite of simulations which demonstrate scalable update latencies, failure-tolerances, message traffic, and resource lookup accuracies. The rest of the paper is outlined as follows. In section 2, we describe our vector representation of resources and their aggregation. In section 3, we present the dissemination of resource information by PDP through the logical hierarchy. Following that, some advanced techniques are discussed in section 4. In section 5, we demonstrate the performance evaluations of PRISM. Related works are discussed in section 6, and we conclude in section 7.

2. Resource Information Description and Aggregation

PRISM features unique resource description and aggregation, which are designed with two major goals: 1) A compact representation of resource information, enabling the scalable aggregation of a large volume of information. 2) Support for multi-attribute queries as well as single-

Node	Vector	Node	Vector
A	(0,3)	*H*	(3,1)
B	(2,4)	*I*	(4,1)
C	(1,2)	*J*	(3,1)
D	(0,3)	*K*	(3,1)
E	(4,3)	*L*	(1,1)
F	(2,0)	*M*	(4,4)
G	(4,3)	*N*	(2,0)

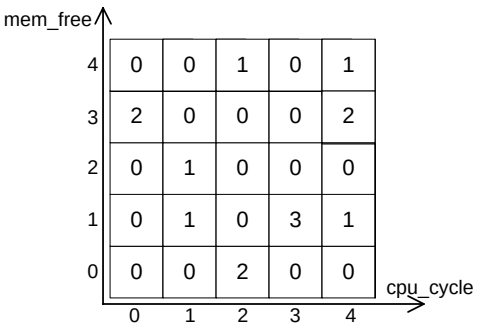


Figure 2: Example of vectors and the aggregated matrix.

attribute ones. To achieve these goals, we introduce the *class* and the *vector* for resource description, and VPA (vector-preserving aggregation), which preserves cross-attribute relations of resource information.

2.1 Class and Vector Representation

Every PRISM node is capable of completing resource lookup locally by maintaining global resource information. Since this information is huge in volume, it may limit scalability. As our solution, the attribute values of resource information are quantized to finite classes, so that the resource information is compactly described with a set of classes, called a vector, rather than with raw values. A class is obtained by applying a predefined 'quantization function' to an attribute value. A vector is an ordered set of *N* classes corresponding to *N* attributes.

Figure 1 illustrates an example of node attribute values, quantization functions, and the resulting vector. For example, by applying the function f_{mem}, the value of the mem_free attribute is mapped to a class '5' (the second element of the vector). Conversely, starting from the vector, one can estimate that the value of the mem_free attribute is between 51200kB and 61440kB.

Note that the quantization functions can be defined in many ways based on application- or attribute-specific properties. The quantization ranges and precisions should be carefully tuned to reflect such properties. We will discuss a case considering the skewed interests of users in Section 4. Further practical designs of quantization functions will be covered in the extension of this work.

The vector representation significantly reduces the size of a node's resource information. In the previous example, a node's resource information consists of four attributes. If each attribute is quantized into 16 classes, each attribute can be expressed with 4 bits. Thus the resulting vector can be represented with only 16 bits, while the original attribute values require a total of 4 integers – 128 bits on 32bit machines. On the other hand, quantizing an attribute value to finite classes inevitably involves some degree of precision loss. However, in shared resource environments, many users simply present the minimum requirements of attribute values to run their ap-

plications. That is, the precision of quantized classes is sufficient for resolving most queries. Even if higher precision is required, the classes can still provide 'hints' for recommending candidate nodes where the exact values can be validated.

2.2 Vector Preserving Aggregation (VPA)

In VPA, the aggregation of multiple nodes' resource information is essentially the 'counting' function of nodes whose vectors are identical. VPA introduces a *matrix* as the expression of aggregated results. Suppose that a node's resource information consists of N attributes. Then the aggregated resource information of multiple nodes is represented as an N-dimensional matrix where each dimension is mapped to each attribute, so that an N-dimensional vector can uniquely locate an entry in the matrix. The aggregation of vectors is thus simply a matter of incrementing the corresponding matrix entries.

Figure 2 shows a simple example of VPA. For ease of visualization, we assume each vector consists of two attributes (cpu_cycle, mem_free) where each attribute ranges over 5 classes. The aggregation results in a 2-dimensional matrix which has 5×5 entries. For example, by simply referring to the value at (3, 1), one can find that there are 3 nodes matching the multi-attribute requirement (cpu_cycle=class 3, mem_free=class 1), demonstrating that VPA preserves the cross-attribute relations of individual nodes. Note that the matrix size is independent from the number of nodes aggregated, allowing VPA to scale to large numbers of nodes.

An N-dimensional array is the simplest way of implementing a matrix. The array requires $O(1)$ cost for search and update. If every attribute has the same number of classes C, then the array holds C^N elements. Alternatively, a tree implementation can be employed for space efficiency. The (vector, count) pairs can be used to form an ordered tree based on vector-hashed keys. The tree can save space by omitting all pairs whose counts are zero. However, with N nodes, search and update costs are higher, $O(\log n)$ and $O(2\log n)$ respectively. Thus, the array implementation is more suitable in the case that updates are frequent and there are a small number of attributes and classes, and the tree in the opposite case.

Regardless of implementation, the matrices are compressed in inter-node exchange to save network bandwidth. The size of raw matrices will be very large. In fact, the matrix size increases exponentially with the addition of one resource metric (one dimension in the matrix). Observing that much of the matrix consists of empty entries, compression can be used to reduce the size of the matrix dramatically. Various well-known compression techniques can be applied (e.g., gzip). The experiment section demonstrates that the matrix size can be reduced to less than 1/100 of its original size.

3. PRISM Architecture

The key design philosophy of PRISM is to preserve the autonomous features of shared resource platforms, thus the PRISM architecture is designed to be decentralized, autonomous, and highly robust against partial failures. These architectural goals are accomplished by *the probabilistic dissemination protocol* (PDP) with *the logical hierarchy*. Section 3.1 and Section 3.2 describe the logical hierarchy and PDP, respectively, followed by a discussion on the resource lookup mechanism of PRISM.

3.1 The Logical Hierarchy

The logical hierarchy defines the relationships among distributed nodes in topology-independent fashion, enabling multiple levels of aggregation and scalable membership maintenance. The logical hierarchy is constructed independently from the physical characteristics of shared resource platforms such as organizational memberships. Note that this independence enables the logical hierarchy to reflect any kind of hierarchical relationships: geographic location, network topology, or organizational structure. In the case of video multicasting applications, the logical hierarchy can be constructed with network topologies.

The key feature of the logical hierarchy is that there are no representative nodes in *any* part of the hierarchy. Figure 3(a) shows an example of the logical hierarchy. The logical hierarchy is basically a tree structure, and all physical nodes (represented as squares in Figure 3(a)) reside at the bottom of the hierarchy as leaves. For identification, each node is given a unique ID from an ID space, similar to the nodes in DHTs [12] [13] [14]. We define a logical unit of membership, the *scope*, (represented as circles in Figure 3(a)). A scope is a set of nodes which share a certain number of digits as a prefix in their IDs. In the example of Figure 3(a), nodes d_{0000}, d_{0001} and d_{0002}, which all have the common ID prefix ‘000’, belong to the scope S_{000}. Similarly, the scopes which share certain prefixes in their IDs belong to an upper-level scope. In Figure 3(a), the scopes S_{000}, S_{001} and S_{002} at level-0 (denoted as L_0) belong to the scope S_{00} at L_1. The same idea is used again at each level through to the top level, L_T. Therefore, the logical hierarchy defines a multi-level membership for each node. The node d_{0110} in Figure 3(a) belongs to the scope S_{011}, as well as to S_{01} and S_0. These scopes define the hierarchical groups from where a node aggregates resource information, and to where a node probabilistically disseminates resource information. The details of the probabilistic dissemination are covered in section 3.2. To explain how resource information is aggregated, we define the following notations for a given scope S.

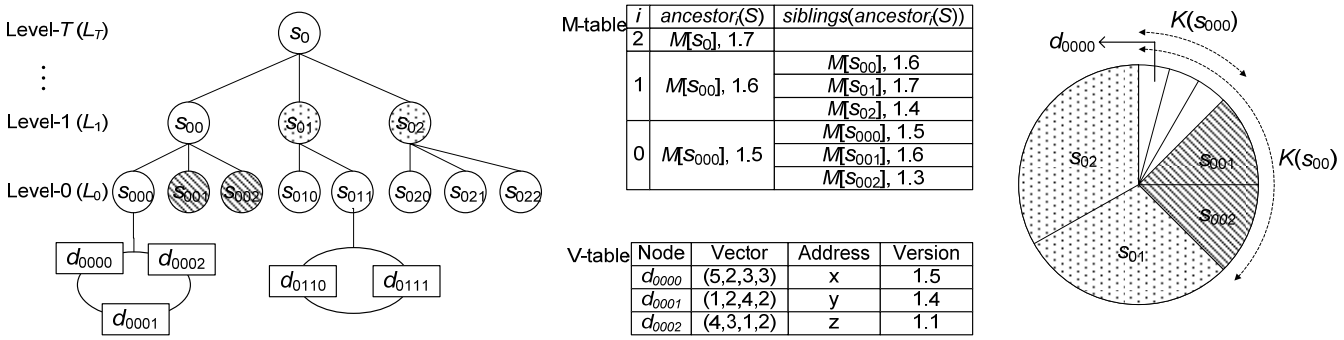


Figure 3: Example of logical hierarchy. (a) Structure of the logical hierarchy, (b) a V-table and an M-table of node *d*1, (c) assignment of DHT key ranges to scopes.

- *parent*(*S*) as the parent scope of *S*
- *siblings*(*S*) as all the sibling scopes of *S*
- $ancestor_i(S)$ as the *i*-th ancestor scope of *S*, thus $ancestor_0(S) = S$ and $ancestor_1(S) = parent(S)$

Resource information is aggregated through the logical hierarchy from bottom to top. At L_0, the vectors of nodes belonging to a scope *S* are aggregated into a matrix, denoted *M*[*S*]. The aggregations of upper levels are similar except that matrices of sibling scopes must be summed. Let S_1, S_2… S_n be sibling scopes at L_k. Then, all matrices of the scopes are aggregated into $M[parent(S_i)]$ by $\Sigma_{i=1\ldots n} M[S_i]$. Exceptionally, at L_0, the vectors are stored along with node addresses and aggregated into a matrix. Since the logical hierarchy reflects the relationship among nodes, the nodes of the same L_0 scope are highly related and are likely to frequently refer to one another. Thus, knowing their node addresses enables direct local lookup at a cost of marginal increase in data size.

Each node retains the vectors or aggregated matrices in the following two data structures:

- *V-table*: A table of vectors for intra-*S* resource information, where *S* is the L_0 scope of the node. The V-table is an enumeration of the unaggregated resource information of all nodes belonging to S.
- *M-table*: A table of matrices for outer-*S* resource information. The M-table is a table of all the matrices of its direct ancestors and their siblings: *S*, $ancestor_1(S)$…$ancestor_T(S)$, and *siblings*(*S*), *siblings*($ancestor_1(S)$)… *siblings*($ancestor_T(S)$), where *T* is the level of the top scope.

Figure 3(b) shows an example of the V-table and the M-table maintained by node d_{0000}. In order to construct and maintain the V-table, a node should obtain the information of nodes in its L_0 scope. For the M-table, the node should get the information of nodes in other scopes. The information of all the nodes is transferred by PDP, which will be explained in Section 3.2.

The logical hierarchy exploits DHT to support self-organizing membership management. In

node joins and leaves, nodes can be self-organized on the DHT substrate. Also, the abstract addressing scheme of the DHT enables a node to be identified by an ID, rather than an IP address. The challenge is how to assign an ID to each node while preserving the logical hierarchy. This is done by assigning a certain range of DHT keys to each scope and assigning a unique node ID from the key range of the scope to each physical node. Figure 3(c) shows the assignment of key ranges to scopes. PRISM assigns a key range prefix whose length is $(L_T–L_i+1)$ to each L_i scope. The key range prefix for a scope S contains those of $ancestor_j(S)$ as its $(L_T–L_i+1–j)$-digit prefixes, so that the key range of a scope includes those of its descendant scopes. For example, $K(S_{00}) = K(S_{000}) \cup K(S_{001}) \cup K(S_{002})$, where $K(S)$ is the key range of scope S. For a node ID, a physical node is given a sequence of digits whose prefix is its L_0 scope's key range prefix, and the remaining digits are filled by hashing its IP address.

3.2 Probabilistic Dissemination Protocol (PDP)

In order to disseminate the matrices in an autonomous and efficient way, we propose a novel dissemination mechanism, PDP. Using PDP, a node decides when and where to disseminate matrices independently. 'When' is controlled by predefined probabilities and 'where' is determined by the logical hierarchy. The novel point of PDP is that even though a node disseminates information through local decisions, all other nodes eventually obtain that information within a configurable period.

PDP is performed in two ways: 1) sharing information among nodes in the same L_0 scope, and 2) disseminating information to nodes of other scopes. In an L_0 scope, a ring topology is formed by each node identifying its predecessor node and successor node. They share their V-tables and M-tables by passing a *token* which contains a copy of the sender's V-table and M-table. When a node receives a token from the predecessor node, it: 1) updates its local tables, 2) recalculates its local M-table, 3) disseminates the matrices of its local M-table to other scopes, and 4) updates and forwards the token.

Step 1 The node replaces vectors and matrices in the local tables if those of the token are more up-to-date. The freshness of vectors and matrices can be determined by examining a version number, i.e. a timestamp. Local clocks can be synchronized using NTP [16], which provides sufficient accuracy[1)] for PRISM.

Step 2 Since the vectors and matrices may have been changed, the node should recalculate $M[S]$ and $M[ancestor_i(S)]$, where S is its L_0 scope, in bottom-up fashion and updates the version numbers of those matrices. For example, $d2$ recalculate matrices in the order of $M[S_{000}]$, $M[S_{00}]$, and $M[S_0]$.

[1)] NTP provides typical accuracies of 1~50ms. [16]

Let n, p^i be the # of child scopes in each level and the dissemination probability for level i, respectively.

Let X be the # of messages sent by a node in a unit time to other scopes.

Let $N = n^T$ be the total # of nodes.

The expected value of X, $E[X] = p_0 n + p_1 n + \ldots + p_{T-1} n = n\sum_{i=0}^{T-1} p_i$, where $p_i = \frac{1}{population} = \frac{1}{n^{i+1}}$

$$\therefore E[X] = n\sum_{i=0}^{T-1} p_i = \sum_{i=0}^{T-1} \frac{1}{n^i} = \frac{1-1/n^T}{1-1/n} = \frac{1-1/N}{1-1/n} < \frac{n}{n-1}$$

Figure 4: The expected number of M-updates.

Step 3 The node tries T probability 'games' for all levels except L_T with a set of predefined probabilities, $\{p_0, p_1, \ldots, p_{T-1}\}$. For only the winning games which are played for L_i, the node sends an *M-update* to each L_i scope except its own L_i scope, where an M-update is a message containing all matrices for L_j ($j \geq i$) in its M-table. The receiving node of each scope is determined by randomly choosing a node id in $K(S')$, where S' is the target scope. The receiving node updates the matrices in its local M-table using the received M-update if its local matrices are out of date relative to those in the M-update.

Step 4 The node overwrites the vectors and matrices of the token with those of its local tables. Then, the node waits for a predefined token holding period (THP) and passes the token to the successor node. THP affects the overall consistency of the system and the amount of network traffic. A shorter THP may cause redundant messages, while a longer one may degrade the system's consistency.

Similarly, the probabilities p_i affect the rate of M-update dissemination, and consequently, the overall consistency and the amount of traffic. Therefore, configuring probabilities is crucial in PRISM. PDP can accommodate any configuration of probability settings. One possible configuration is to set a scope's probability inversely proportional to the node population of the scope. This configuration implicitly offers equally expected rates of dissemination from each scope. Figure 4 shows the analysis of the number of M-updates which are sent by a node per unit time. It shows that although we increase the number of nodes by increasing the number of levels, the number of M-updates remains bounded by n / (n-1). PDP can assign different probabilities to different levels of the logical hierarchy to adjust the rate of dissemination.

3.3 Resource Information Lookup

Information sharing of PRISM enables simple and fast resource information lookups. Since a node maintains resource matrices of other scopes, it can handle most resource queries locally. Assume that an example query is "SELECT n resources FROM nodes WHERE CPU > 2GHz CPU, available memory > 512MB, bandwidth > 100 MB". First, the requirement is translated into a vector such as (2, 2, 1). The node then scans its local V-table. If n or more resources in its L_0

scope satisfy the requirements, the lookup is completed with this local search. Otherwise, the node checks $M[siblings(ancestor_i(S))]$ in its local M-table for sufficient resources, incrementing i from 0 to T. If available resources are found in $M[S']$, the node forwards the query to any node in scope S'. The receiving node then repeats this process.

Due to update delays, the vectors and the matrices may not be absolutely correct. Thus, PRISM should verify if discovered resources actually satisfy the requirements. We call this process *verification* and the failure of verification a *miss*. If verification fails, PRISM finds other resources that satisfy the requirements and supplements the results. The miss rate is dependent on update delays governed by the dissemination probabilities. However, due to the trade-off between the overall consistency and the amount of traffic, we cannot increase the probability to reduce the miss rate without cost. Instead, as a novel solution, more nodes can be requested than originally required. Although a few nodes fail verification, enough nodes may pass with high possibility. In this way, PRISM can reduce supplementary lookups and respond to queries quickly.

4. Advanced Techniques

In this section we present techniques that refine PRISM in order to improve performance and efficiency. We offer compensation techniques to enable better control over PDP, and Non-Uniform Quantization to reduce message size and provide users with finer-grained information.

4.1 Compensation Techniques

Probabilistic dissemination is inherently uncertain. For instance, an L_0 scope's resource information may not be disseminated for a long period of time, thereby increasing the propagation delay. It is also possible that M-updates will be disseminated too frequently, increasing network traffic while contributing little to the overall freshness of resource information with redundant M-updates. Finally, it is probable that the distribution of successful probabilistic trials will be non-uniform. With a probability of $1/N$, one may expect the occurrence of M-updates to be every N trials. However, this expectation is not always fulfilled. We present here two compensation techniques that address these three undesirable situations.

4.1.1 Update Frequency Compensation

Update Frequency Compensation provides solutions to the first two situations: that of infrequent dissemination and of overly frequent dissemination of M-updates. For a given time interval $N \times$THP, the number of M-updates follows a binomial distribution. It can be approximated to a normal distribution with $mean = Np$ for a given probability p. This distribution ranges between zero ~ N (assuming one trial per unit time). Obviously, zero, no successes, is not desirable. Con-

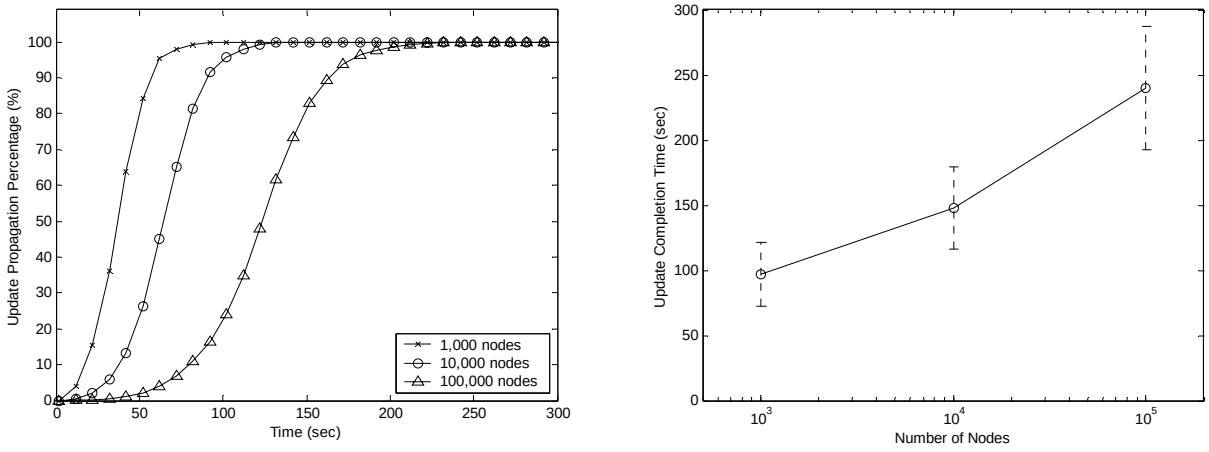
versely, *N*, all successes, might be too frequent. Values close to zero and close to *N* are similarly undesirable. We compensate for these extremes in probabilistic dissemination by modifying PDP. We mark the last M-update timestamp on the token and examine whether or not the elapsed time since the last M-update is greater than a predefined time interval. If such a time-out occurs, an M-update is forcibly sent, preventing infrequent dissemination. To compensate for the overly frequent case, we check if there has been any M-update in the last *k* trials (where *k* is a user-defined parameter) before performing the current probabilistic trial. This ensures that M-updates are not disseminated more often than every *k* THPs. By applying simple modifications to the token passing process, our solution assures that the infrequent and overly frequent cases can be prevented.

4.1.2 Uniformity Compensation

Uniformity Compensation solves the third undesirable situation, providing an adaptive solution so that the distribution of the number of trials is mostly concentrated around the mean. For example, a probability setting of p=0.1 implies that it is preferable to disseminate M-updates roughly once every ten trials. In practice, however, the number of binomial trials between successful trials follows a geometric distribution. We dynamically adjust the probability parameters in order to obtain a distribution mostly concentrated around the mean, which is ten trials for p=0.1 That is, the probability setting is adjusted to increase or decrease the probable frequency of a successful trial. For example, if the last success was ‘late’, the probability will be increased in order that following successes occur ‘earlier’. The probability is incremented in proportion to the number of trials since the last success. That is, the longer it’s been since the last M-update, the more we increase the probability. However, if M-updates are being disseminated too frequently, we reset the probability back to a minimum value.

4.2 Non-Uniform Quantization

Non-Uniform Quantization provides variable levels of granularity for resource classes. As users are typically more interested in under-loaded nodes, we aim to provide finer quantization for higher user-interest regions. On the other hand, dividing the range of resources into too many classes will increase the message size of updates. Our solution is to divide the space of possible classes into non-uniform sized divisions: finer-grained classes are used to categorize high levels of free resources and course-grained classes for fewer available resources. With Non-Uniform Quantization, users are able to query classes that they have greater interest in at a fine granularity. While this would normally increase the size of messages, this increase can be offset by the decrease in message size afforded by course-grained quantization of the low-interest classes.



(a) (b)

Figure 5: Update propagation delay; (a) the update progress over time, (b) update propagation completion latencies.

5. Performance Evaluation

5.1 Environment Settings

We implemented an event-driven simulator in JAVA, running on top of the transit-stub topology generated by the GT-ITM topology generator [19]. We performed simulations varying the number of nodes from 10^3 to 10^5. For each setting for the number of nodes, the logical hierarchy is constructed as a balanced tree with degree 10. We set the token holding period (THP) in L_0 scopes to one second. A scope's PDP probability is configured as $1/(\textit{the node population of the sphere})$ as discussed in Section 3.2. For simplicity, we assume local resource information consists of five attributes, each of which is quantized to five classes.

5.2 Experimental Results

5.2.1 Update Propagation Delay

We begin by measuring single update propagation delay, the time taken for an update of a node to be propagated to all nodes, which is critical to the overall consistency of PRISM. Figure 5(a) shows the percentage of update propagation completed from the time when the update was initiated. For even the maximum number of 10^5 nodes, the single update propagation delay is less than 240 seconds. This suggests that a change in resource status of a node can be completely propagated to all nodes before subsequent changes under practical conditions[1)]. Figure 5(b)

[1)] A previous study [8] reported that 94% of nodes hardly change their resource status for 5 minutes.

shows the average and the 95th percentiles of the single update propagation delay according to the number of nodes. The delay increases logarithmically with the number of nodes. This is because the update is further propagated by the receivers; therefore the number of updated nodes grows exponentially over time. This demonstrates the scalability of PDP for a large number of nodes in terms of update propagation delay.

5.2.2 Overall Consistency

Due to dynamic changes of resources, a node's V-table and M-table may contain stale resource information. Such staleness influences the accuracy of resource lookups, resulting in a decrease in lookup performance. We define the *freshness* of a node's *k*th-level matrix as follows.

$$freshness_k = \left(1 - \frac{\sum_{j=1}^{m} |a_j - b_j|}{2n}\right) \times 100\%$$

a_j : the jth value of L_k matrix on a node's O-table
b_j : the jth value of up-to-date L_k matrix
n : the number of nodes
m : the number of values in the matrix, $(\#\text{ of classes})^{(\#\text{ of attributes})}$

Freshness is essentially a measure of how identical a local matrix is to an up-to-date matrix (which is maintained by our simulator for measurement purposes only). Note that freshness is defined and measured per level because the information accuracy of matrices is dependent on the level of the logical hierarchy. In addition to the logical hierarchy, the frequency of resource changes greatly affects the freshness of matrices; the more frequent the changes, the less fresh the matrix. For our experiments, we introduce the *average resource change interval* (ARCI), the average interval over which nodes' local resources change; a lower ARCI implies that resources change more frequently.

Figure 6 shows the freshness values for scopes at L_0, L_1, and L_2 along with ARCIs of 2, 5, 10, and 15 minutes. The graphs show higher freshness can be achieved with a longer ARCI. Also, freshness is higher at lower levels of the logical hierarchy. This proves that PRISM provides higher information accuracy for more closely related nodes. As in a previous study [8], we can assume an ARCI of 5 minutes to be practical, where all per-level freshness exceeds 90%. Thus, under these realistic conditions, no more than 10% of information maintained by PRISM is stale.

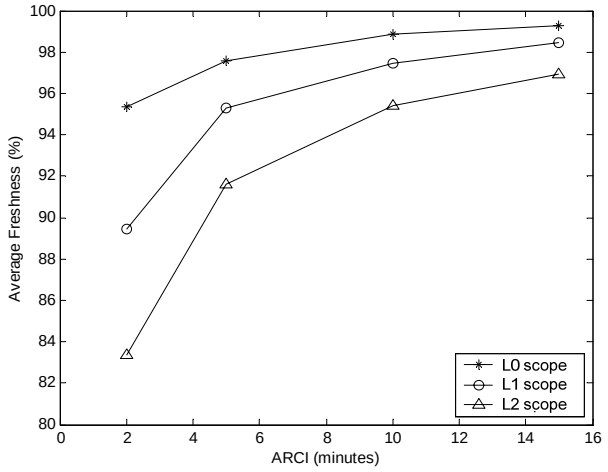


Figure 6: Average freshness with ARCIs

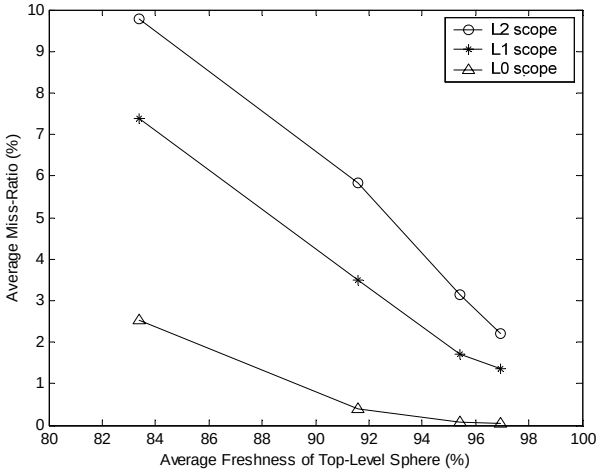


Figure 7: Average miss-ratio with freshness

5.2.3 Lookup Accuracy

Freshness influences the performance of resource lookup. As mentioned in Section 3.3, there may be *miss*es during the verification phase due to stale information. The misses require additional lookups, thereby increasing the response time. As a measure of lookup efficiency, we define *miss-ratio* as follows.

$$Miss\text{-}ratio = \left(\frac{\#\ of\ misses}{\#of\ verifications} \right) \times 100\%$$

Figure 7 shows the miss-ratio of resource lookups under various freshness values of L_{top} scopes. With a freshness of 91.5% resulting from an ARCI of 5 minutes, we obtain miss-ratios of 0.4%, 3.4%, and 5.8% for L_0, L_1, and L_2, respectively. In other words, there is a probability of 5.8% that additional lookups are required when finding resources at L_2. All these results support our assertion that PRISM can discover necessary resources quickly.

5.2.4 Robustness

Figure 8 demonstrates the robustness of PRISM. By forcing 20% of nodes to fail at time=300 sec, we cause a significant change in the global resource information. Due to the update propagation delay, this change inevitably degrades overall freshness. We can observe that the freshness drops instantly at time=300 but recovers within about 50 seconds as the change is propagated. This demonstrates the robustness of PDP; it continues disseminating M-updates without being affected by node failures.

5.2.5 Message Traffic

In this section, we evaluate the overall message traffic of PDP. As discussed in Section 2 and

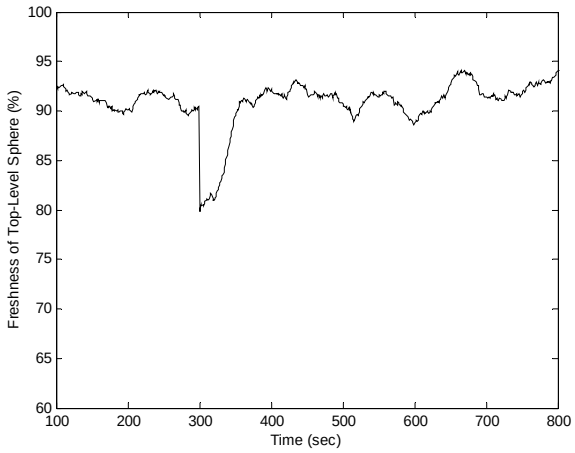


Figure 8: Robustness.

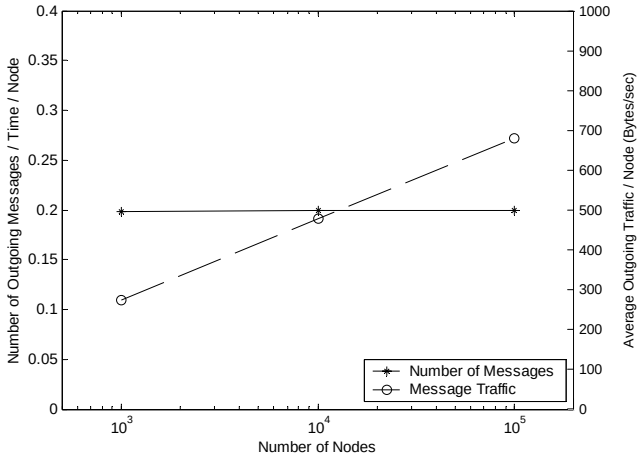


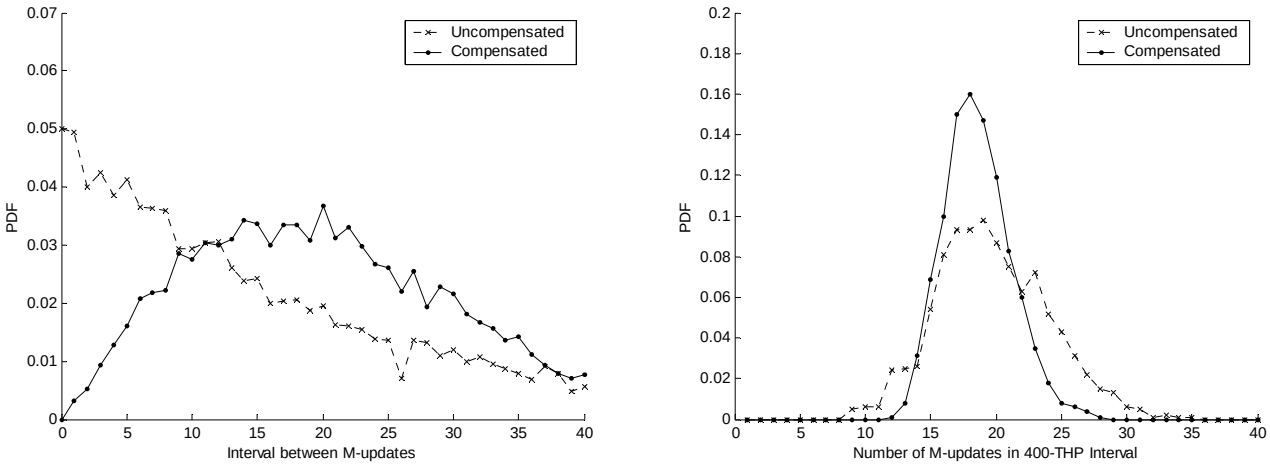
Figure 9: Message traffic.

3, the size of a message is dependent on the number of resource attributes and classes as well as the level of the destination scope in PDP. For a compression technique as mentioned in Section 2, we apply a simple gzip compression onto the array implementation of matrices. The result is that the compression ratio ranges from 101:1 to 163:1 and the size of the compressed message from 2.03KB to 6.50KB. Such a high compression ratio is mainly due to the high redundancy of empty entries in the matrix. Figure shows both the number of messages and the volume of traffic sent by a node per unit time. The number of messages is nearly constant as n increases, while the traffic grows O(log n), where n is the number of nodes. The results are consistent with our analysis in Figure 4, so that the growth is hardly noticeable when we vary the number of nodes from 10^3 to 10^5. Also, the traffic increases linearly with the number of hierarchy levels, N, because each message contains an M-table with up to N levels of matrices. This shows that PRISM generates a feasible volume of traffic which grows in scalable way as the number of nodes increases.

5.2.6 Effect of Compensation Techniques

To validate the compensation techniques proposed in Section 4, we place 20 nodes in a L_0 scope and set the probability p_0 to 1/20, which implies 20 seconds as the desired interval between M-updates. Figure 10(a) shows that while the uncompensated case results in a geometric-like distribution, using uniformity compensation, the distribution of the intervals between M-updates becomes bell-shaped with the mean around 20 seconds.

Update frequency compensation prevents both infrequent and overly frequent dissemination. As the parameters for update frequency compensation, we set the time-out to 40 seconds and k to 2, the history length to check. Figure 10(b) is the number of M-updates observed for a 400-THP interval, in which around 20 (equals to $400p_0$) M-updates are desired. The compensated result shows a much narrower range, approximately (11, 28). The uncompensated case, however, is



(a) (b)

Figure 10: Compensation techniques; (a) uniformity compensation, (b) update frequency compensation

more broadly distributed.

6. Related Work

Much work has been done in resource discovery systems for shared resource platforms like Grid [10], XenoServer [11], and PlanetLab [15]. Here, we summarize some resource information systems: Metacomputing Directory Service (MDS) [1] for Globus [2] (the most well-known architecture for the Grid), XenoSearch [5] for XenoServer, and SWORD [8] for PlanetLab.

Metacomputing Directory Service (MDS) uses a hierarchical architecture to manage globally distributed resources. The two major components of MDS, Grid Resource Information Service (GRIS) and Grid Index Information service (GIIS) provide access to dynamic resource information and its aggregation, respectively. The hierarchical architecture and manual configuration of MDS, however, limit its robustness and scalability.

To resolve these shortcomings, [3] and [4] proposed self-organizing approaches by adopting P2P mechanisms, Gnutella [21] and Pastry [13], respectively. Although they use different P2P networks, their operation are functionally the same. A local information manager stores the resource information of its local resource pool and acts as a peer to other information managers in the P2P network. While Gnutella and Pastry can handle dynamic joins or leaves of peers automatically, queries are flooded to all the peers in the overlay in order to obtain resource information. The query flooding leads to network and system overload, restricting the scalability of these systems.

XenoSearch and SWORD handle multi-attribute resource information using DHT. They map the values of attributes as input keys into DHT key spaces. While XenoSearch uses a separate

DHT overlay for each attribute, SWORD divides the DHT key space into as many regions as the number of attributes. The major difference between these systems and PRISM is the lookup method; while SWORD and XenoSearch need to scan all remote nodes responsible for the requested value range, PRISM can resolve most queries with locally stored summary information. Moreover, these systems may suffer from hot-spots over the key ranges which users are interested in.

Some studies have proposed distributed information management systems which may be applied to deal with resource information. SDIMS [7] and Cone [9] use aggregation for scalability like PRISM. They are different from PRISM in that each physical node maps to a node in the aggregation trees. Thus, these systems are not resilient to the failure of nodes, in particular the root node. On the other hand, the aggregation tree of Astrolabe [6] is not physical but logical in that all nodes share the information of interior nodes. Thus, it is robust against partial failures. Also, Astrolabe uses a gossip protocol to propagate information through the aggregation tree in an autonomous fashion like PDP. However, it cannot adjust the gossiping rate for distinct levels in the tree, so that it does not support multi-levels of information dissemination considering the relationship among the nodes. Additionally, since Astrolabe has a local group representative which is responsible for data dissemination, PRISM is more robust than Astrolabe against partial failures.

7. Conclusion

In this work, we have presented PRISM, a fully distributed, autonomous, and highly robust system for resource information management in shared resource platforms. We presented the *class*, the *vector* and the *matrix* for resource description along with our VPA (vector-preserving aggregation) in order to achieve the goals of scalable aggregation and of support for multi-attribute queries. The Probabilistic Dissemination Protocol (PDP) enables autonomous and robust dissemination of resource information, achieving scalability and traffic-efficiency. We have demonstrated the validity of our approach through simulation by testing the scalability of update propagation, consistency, failure-tolerance, resource lookup accuracy and message traffic.

There are a number of on-going extensions of PRISM. It is now under deployment on PlanetLab for validation in real environments. We exploit FreePastry [23] and Ganglia [24] as the DHT substrate and the resource monitoring tool, respectively. Along with the real deployment, we are building a concrete mathematical analysis of PDP to ensure a high degree of confidence in its probabilistic operations. In addition, we are resolving the problem of dynamicity of the logical hierarchy in the case of frequent node joins and leaves. The quantization scheme is also being extensively studied to demonstrate its practical significance on real resource metrics. We envision

applying the concept of PRISM onto newer applications such as information discovery in ubiquitous environments.

References

[1] Karl Czajkowski, et al. Grid Information Services for Distributed Resource Sharing. In *Proc. 10th IEEE International Symposium on High-Performance Distributed Computing,* 2001

[2] I. Foster, C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl J. Supercomputer Applications*, 11(2):115-128, 1997.

[3] Adriana Iamnitchi and I. Foster. On Fully Decentralized Resource Discovery in Grid Environments. In *Proc. 2nd International Workshop on Grid Computing*, 2001.

[4] Ali Raza Butt, Rongmei Zhang, and Y. Charlie Hu. A Self-Organizing Flock of Condors. In *Proc. ACM/IEEE conference on Supercomputing*, 2003.

[5] David Spence and Tim Harris. XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform. In *Proc. 12th IEEE Symposium on High Performance Distributed Computing*, 2003.

[6] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems*, Vol. 21, No. 2, May 2003, 164-206.

[7] Praveen Yalagandula and Mike Dahlin. SDIMS: A Scalable Distributed Information Management System. In *Proc. of the ACM SIGCOMM Conference*, 2004

[8] David Oppenheimer, et al. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. *14th IEEE Symposium on High Performance Distributed Computing,* 2005.

[9] R. Bhagwan, P. Mahadevan, G. Varghese, and G. M. Voelker. Cone: A Distributed Heap-Based Approach to Resource Selection. Technical Report CS2004-0784, UCSD, 2004.

[10] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[11] K. A. Fraser, et al. The XenoServer computing infrastructure. Technical Report UCAMCL - TR-552, University of Cambridge, 2003.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of the ACM SIGCOMM Conference*, 2001.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms* (Middleware), November, 2001.

[14] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of SIGCOMM*, Aug. 2001.

[15] Planetlab. http://www.planet-lab.org

[16] RFC 2030 (SNTP V4). http://www.ietf.org/rfc/rfc2030.txt

[17] KaZaA. http://www.kazaa.com

[18] Akamai Technologies Inc. http://www.akamai.com

[19] E. W. Zegura, et al. How to Model an Internetwork. In *Proc. of IEEE Infocom*, 1996.

[20] The Human Proteome Folding Project. http://www.grid.org/projects/hpf/

[21] Gnutella. http://www.gnutella.com

[22] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, 2004.

[23] FreePastry. http://freepastry.rice.edu/FreePastry

[24] Ganglia. http://ganglia.info/