Highly Available Component Sharing in Large-Scale Multi-Tenant Cloud Systems

Juan Du, Xiaohui Gu, Douglas S. Reeves
Department of Computer Science
North Carolina State University
jdu@ncsu.edu, {gu,reeves}@ csc.ncsu.edu

ABSTRACT

A multi-tenant cloud system allows multiple users to share a common physical computing infrastructure in a cost-effective way. Component sharing is highly desired in such a shared computing infrastructure, where different tenants can leverage each other's information and expertise to fulfill their own tasks. However, it is challenging to maintain the availability of sharable component resources in a large-scale cloud infrastructure, as cloud tenants are fully autonomous and highly dynamic. In this paper, we present a novel highly available component sharing system for large-scale multi-tenant cloud systems. We describe a component availability prediction scheme to identify endangered components (i.e., components at risk of extinction) within the infrastructure. The system then performs predictive replication based on the availability prediction results to preserve those endangered components. Thus, our system can preserve the availability of all component resources with low cost. Theoretical analysis and large-scale simulation are used to quantify the accuracy of our component availability prediction, and the efficiency of predictive replication. Experimental results show that our scheme can predict endangered components with high accuracy, and achieve up to 99% availability with about 15% of the full replication cost.

1. INTRODUCTION

Multi-tenant cloud systems [1,2] have recently emerged as promising shared computing infrastructures. Cloud users, called tenants, can lease computing resources in a pay-as-you-go fashion. In contrast to dedicated computing systems, cloud systems offer a more cost-effective solution, since tenants can perform various computing tasks without maintaining expensive physical computing infrastructures themselves. Resources can be dynamically provisioned or released based on the tenant's needs and schedule. Cloud systems can be leveraged by many resource-intensive scientific computing applications that often require a large number of computers for performing experiments and analyzing data [13].

Component sharing is highly desired in shared cloud computing systems when different tenants want to leverage each other's information and expertise to fulfill their tasks. We use the term "components" to generally refer to sharable objects, such as data files and reusable software components. For example, a scientist can lease a hundred nodes from the cloud and download ten years of world temperature information to perform a global warming trend analysis. The analysis result becomes a sharable component that can be leveraged by other scientists to perform further studies. In contrast to dedicated computing systems, a cloud tenant can release resources in the middle of a computation, and resume leasing later. To avoid redundant computation, it is important for the cloud to provide a component preservation service that allows tenants to store data that can be used later by the same tenants, or by other tenants. Similarly, tenants may want to share reusable software components such as data mining libraries. However, cloud computing typically grants full autonomy to its tenants to download or delete components arbitrarily. Moreover, frequent tenant departures or arrivals induced by the charging model of the cloud system make it challenging to provide a scalable and highly-available component sharing system.

To preserve component availability, a simple approach might be to deploy a centralized component manager to maintain a backup copy of all unique components in the system at all times. However, this simple approach has three fundamental problems. First, there may be thousands of tenants and many more components in the system. For scientific computing applications, data objects often have large sizes that can range from hundreds of Megabytes to Gigabytes. Thus, component sharing will incur a prohibitive storage cost on the management node. Second, tenants can add or delete components arbitrarily. It is very costly to maintain an updated global index table to track all unique components in the infrastructure. Third, the centralized approach has the problems of single point of failure and scalability limitation.

Although structured peer-to-peer (P2P) systems [4, 9, 14, 21] can easily achieve availability by maintaining a fixed number of components, it is hard, if not impossible, to enforce and maintain a uniform structure over large-scale multi-tenant cloud infrastructures. Particularly, structured P2P systems need to control the placement of different components, which may not be accepted by autonomous tenants. Existing replication schemes for unstructured P2P systems [8] mainly focus on enhancing search and download performance, rather than preserving component availability.

In this paper, we present a novel highly available component sharing system for multi-tenant cloud systems. One major objective of our approach is to maintain the autonomy of cloud tenants while preserving availability of all components at all times with minimum storage cost. To reduce storage cost, our system selectively replicates a subset of components to preserve component availability. The intuition behind this approach is that it is unnecessary to replicate those components that already have a lot of copies on different tenants. Tenants can download such "popular" components from each other directly without involving cloud management nodes [1].

[1]We assume in this paper that a mechanism is in place to provide incentives for tenants to share components with each other.

HPDC'10 June 23–25, 2010, Chicago, Illinois.

However, the challenging task is to identify which subset of components should be replicated on the management nodes while preserving the availability of all components. To address the problem, we develop a component availability prediction scheme that can identify, in a timely way, *endangered* components that have few copies in the system, and are at high risk of extinction. Thus, instead of proactively replicating all components as in the simple approach, our scheme aims at replicating only these endangered components. This allows a significant reduction in the storage cost.

It is a challenging task to dynamically identify endangered components without relying on a centralized server to track component copies, or to enforce component placement rules (as in structured P2P systems). Furthermore, component availability status may vary over time as the demand for different components changes. For example, some endangered components may become highly available as more tenants start to download them. Thus, the availability management system must be adaptive. To address the challenges, we propose a distributed hybrid component availability prediction algorithm. Our scheme first leverages the component query results to remove popular components from the list of endangered components. The intuition behind our approach is that if a component is frequently searched by different tenants, it will naturally have many copies on different tenants. Thus, the availability management system does not need to worry about extinction of those popular components. For those components that do not receive many recent queries, we employ a dynamic random walk sampling scheme to estimate the number of replicas, and predict endangered components.

Based on the availability prediction results, our system can dynamically replicate endangered components on dedicated management nodes to prevent component extinction. In a large-scale shared computing infrastructure, our availability management scheme reduces unnecessary replication of popular components and guarantees the availability of endangered components with limited replication cost. Our component management scheme is generic, which can be integrated with existing cloud storage systems such as Amazon Simple Storage Service (S3) [12].

We have implemented a prototype of the highly available component sharing system and conducted a large-scale simulation study. Our experimental results reveal several interesting findings. First, the proposed scheme can accurately identify endangered components and maintain high availability in a highly dynamic multi-tenant cloud system. Second, our scheme can achieve up to 99% availability with about 15% of full replication storage cost. In contrast, other selective schemes can achieve similar or 10-15% lower availability with more than 30% more storage cost. Third, our scheme is light-weight, which imposes little computation and communication cost to the cloud system.

The remainder of this paper is organized as follows: Section 2 introduces the system model. Section 3 presents the design of our highly available component sharing system in detail. Section 4 presents the experimental results. Section 5 compares our work with related work. Finally, the paper concludes in section 6.

2. SYSTEM MODEL

Our component sharing system adopts a two-tier hierarchical structure for scalability and easy deployment, which is illustrated by Figure 1. The system consists of two types of nodes: 1) *tenants* (a_i) that are autonomous cloud users, and 2) *component managers* (CM_i) that are dedicated infrastructure nodes. Table 1 summarizes all the notations used in this paper. The component managers are responsible for maintaining the availability of all components owned by the infrastructure. Each component manager connects to a set of tenants and also maintains connections to some other component managers.

Each component manager keeps local index tables for all components used by its connected tenants. The index table provides the information about the number of replicas and locations of replicas for each unique component. Since a component manager only connects to a limited number of tenants, the storage cost for the index table is very low. The component manager also monitors the status of those tenants connected to it, such as join, leave as well as component addition and deletion. When a tenant joins or leaves the system, or adds or deletes a component, it sends a corresponding update message to its local component manager. The component manager then updates its index table accordingly. The maintenance cost for updating the index table is low compared to the cost of queries, because such update messages are much less frequent than queries.

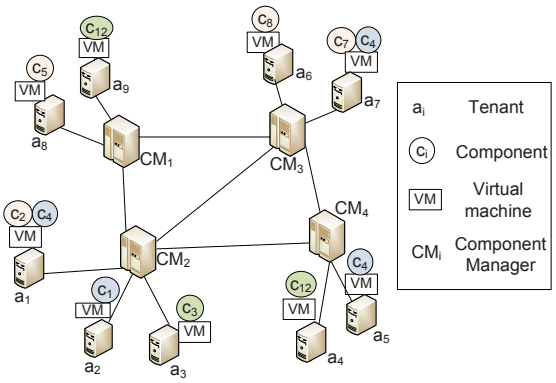


Figure 1: System architecture.

notation	meaning
CM	Component manager
c_i	Unique component
a_i	Tenant
N_f	# of unique data components in the system
N_p	# of component managers in the system
r_i	Popularity rank of the ith component
d	Avg # of tenant connections at CM
ω	Avg walker length
k	# of random walkers
t	Replication threshold
R	Repair ratio

Table 1: Notations.

3. SYSTEM DESIGN

In this section, we present the design details of our highly available component sharing system. We first provide an overview of our approach. We then describe our component availability prediction algorithms, followed by an analytical study to reveal the theoretical underpinning of our approach.

3.1 Approach Overview

We now describe the basic component sharing protocol, illustrated by Figure 2. When a tenant a_i creates a new sharable component, it first registers the component with its local component manager CM_i. When another tenant a_j would like to download the component, it sends a component query to its local component manager CM_j. The component manager CM_j first checks whether the requested component is available on its connected tenants based on the local component index table. If the component is unavailable within the local domain, CM_j further propagates the component query to other component managers. The query is associated with a time-to-live (TTL) counter to indicate the number

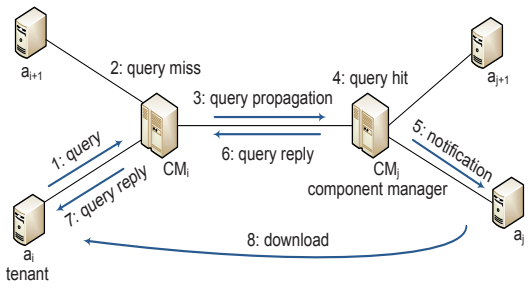


Figure 2: Component sharing protocol.

of hops it can traverse. The query is propagated among different component managers based on the TTL counter. All component managers that receive the query and have the requested component will reply to the query with the locations of the tenants that have the component. The tenant a_i can then download the component from the discovered tenants. For fault-tolerance purpose, the system may return multiple tenants that hold the requested component and perform parallel downloading from multiple tenants. Essentially, the component managers serve as the middlemen among tenants to efficiently share components. The basic component sharing protocol follows the design of real world super-peer file sharing architectures such as Gnutella [3].

If component managers only store component indexes, the availability of a component will purely depend on whether a tenant still holds the component, and whether the same tenant is still leasing resources from the cloud. However, as mentioned in the introduction, tenants can freely leave the cloud or delete a component. Some components, especially those "unpopular" components that have few copies in the system, will be at risk of becoming completely unavailable when all copies are removed from the system, which is termed component extinction. To achieve highly available component sharing service, it is necessary to replicate components on the component managers to preserve their availability. However, component managers typically have limited storage capacity, and it could easily be impossible to back up all components at all times. To provide a practical highly available component sharing system, it makes more sense to selectively replicate components in danger of extinction, rather than replicating all components.

For this purpose, a dynamic component availability prediction scheme identifies endangered components. Without a centralized component tracking server, it is challenging to identify endangered components, since they have very few copies in a large-scale system. To address this challenge, we adopt an elimination-based component availability prediction approach. Our component availability prediction scheme tracks component availability by first eliminating non-endangered components locally and then conducting a dynamic system-wide availability investigation for further elimination.

Specifically, in the first step, component managers leverage component queries to identify local endangered components. Component managers monitor queries for locally stored components, and identify local *non*-endangered components by marking those that receive frequent query requests from different tenants. The intuition behind our approach is that a new component copy will be instantiated on a tenant after a successful component query. Thus, if a component is frequently queried by different tenants, it will naturally have many copies on different tenants.

In the second step, component managers check the availability of local endangered components in the maintained index table, and if necessary, conduct further elimination through a system-wide availability investigation. If enough copies of a component are found in the index table, it indicates that it is not an endangered component. Otherwise, our scheme employs a dynamic random walk sampling mechanism among component managers to estimate the system-wide component availability.

```
ComponentManager(msg)
1.  if (msg.type == tenant leave or component delete)
2.      mark local endangered component(s) c_i
3.      if (msg.tenant has c_i)
4.          for each component c_i
5.          if (system record is valid and sys.count ≥ t)
6.              c_i is non-endangered
7.          else if (local record is valid and local.count ≥ t))
8.              c_i is non-endangered
9.          else
10.             generate new local record from index table
11.             if (local.count ≥ t)
12.                 c_i is non-endangered
13.             else
14.                 send popularity query through random walkers
15.if (msg.type == popularity query)
16.     put valid system record and local record in the walker
17.     if (valid sys.count ≥ t, or valid local.count ≥ t)
18.         send popularity reply to initiating CM
19.     else if (not last hop and has non-visited neighbors)
20.         randomly select a CM neighbor to forward
21.     else
22.         send popularity reply to initiating CM
23.if (msg.type == popularity reply)
24.     if (all walkers are back or timer expired)
25.         generate new system record
26.         if (sys.count <t)
27.             mark c_i as endangered
```

Figure 3: Pseudo code of highly available component management algorithm.

Based on the availability prediction results, we dynamically replicate endangered components on component managers to prevent component extinction. Figure 3 shows the pseudo code of the highly available component sharing algorithm. Note that our approach is fully distributed at the component manager level, which does not assume a global component index table.

3.2 Component Availability Prediction

Our scheme identifies endangered components using a distributed hybrid approach. First, each component manager marks a set of local components as endangered based on all recently received queries. Second, for such components, component managers further investigate the local availability, and eliminate non-endangered components by conducting system-wide investigation using random walks.

Local endangered component identification. Component managers leverage component queries to propose local endangered components. Intuitively, a component that is queried more often should have more copies in the system. Each component manager is responsible for monitoring component queries received from other component managers as well as its connected tenants. A component manager can then rank the local components stored on its connected tenants according to query frequencies, and may label some of the components as endangered based on its independent query monitoring.

Each component manager maintains a "LFQ" (Least Frequently Queried) queue for components provisioned by its connected ten-

ants, and adjusts the component rank whenever receiving a query. As a result, frequently-accessed components gradually take up the front positions of the queue while the tail of the queue is left to seldom- or never-queried components. For each component manager, the same fraction of components at the tail are marked as local endangered components. Thus, components for which there are frequent queries are largely excluded at this step.

System-wide component availability investigation. If a component manager concludes there is an insufficient number of replicas of a component, based on query monitoring results, random walk sampling is triggered to further validate that the component is indeed endangered. The component manager further eliminates non-endangered components through estimating component popularity. This is achieved by aggregating information from other component managers.

The system-wide component availability investigation involves a dynamic random walk sampling process among the component managers. To minimize the frequency of random walks and cut down the system cost, each component manager maintains a limited amount of component state information in local storage. If the state information can indicate the component has enough replicas, the component is eliminated from the list of endangered components. Thus, the infrastructure does not need to replicate the component. Otherwise, the dynamic random walk sampling is conducted among component managers to estimate the number of replicas system-wide.

Component managers maintain two records about the state of components in the system. One is called the *local record*, representing the component population across the tenants that connect to the component manager. Since component managers keep indexes for every component on connected tenants, the local record is accurate and complete. The other state information is called the *system record*, containing information about the component population that is collected through random walks.

At each component manager, local record is computed for every component that resides in the tenants connected to this component manager, and is used to keep count of the number of local copies. System records are kept for all marked local endangered components, to keep count of the number of remote replicas found by random walks. Each record is attached with a timestamp and expires periodically. Local records are updated each time a component manager receives join, leave, and update messages from connected tenants. System records are updated from the random walk results, which will be described below. If either local record or system record for a component shows that its replica count exceeds a pre-defined threshold, the component is regarded as having sufficient copies, and is therefore not endangered. Otherwise, if the system record indicates the component population falls below the threshold and the record for the component has not expired, the component is identified as being endangered.

A component manager initiates several random walkers containing information about the component to be investigated, and sends them to randomly selected component manager neighbors. These neighbors are responsible for storing their own up-to-date statistics about this component in the walker, and in turn, forward the walker to their own randomly selected neighbors. The above statistics include both the local record and system record associated with the component, as long as the records have not expired. Walkers may be terminated when enough replicas are found, or after a fixed number of hops, if not all neighbors have been visited at any intermediate node. The component manager where a walker terminates sends the result back to the initiating component manager.

In the proposed method, the random walk process uses each of the visited component managers along a walker as a sample point, rather than sampling the end points of each independent walker. These two methods achieve similar statistical properties, according to [10]. However, by sampling visited nodes instead of end nodes, the scheme demands much less number of walkers to achieve the same sample size. To ensure no individual walker visits the same component manager twice, a message ID field is included in the message header. Component managers store recently seen IDs and terminate a walker if its message ID has already been seen.

We now justify that random walk sampling results will follow the actual component popularity distribution. We define the popularity of the ith -ranked component, denoted by r_i, as the fraction of component managers having a replica of component c_i, where $0 \leq r_i \leq 1$. If we assume component replicas are randomly distributed among all tenants, then r_i is also the probability of a tenant having a replica of the component c_i. Each component manager has aggregated information from d tenants. Thus, the number of replicas of component c_i found via a single random walker has expected value $\omega \cdot d \cdot r_i$, where ω denotes the walker length.

The random walk sample size, corresponding to the product of walker length ω and number of walkers k, is a heuristic and tunable factor that affects the estimation accuracy. A small sample size can incur biased estimation, while a large sample size may lead to higher system cost and delay. Its impact can be evaluated by the degree of false negatives and false positives. In the following, we show a method to appropriately determine random walk sample size.

Component popularity takes the range of [0,1]. We use two numbers r_l and r_h to classify components as endangered or non-endangered, where r_l and r_h denote the popularity of the lth and hth rank components respectively. We define that a component c_i is endangered if $r_i < r_l$, or is non-endangered if $r_i > r_h$, where $0 < r_l < r_h < 1$. We assume both r_l and r_h are given. The range $[r_l, r_h]$ is a grey zone with a mean $r_t = (r_l + r_h)/2$. We set r_t to be the threshold popularity. Given r_t and sample size $\omega \cdot k$, the expected number of replicas found in the sample is $t = \omega k d r_t$ representing the replication threshold. Whenever the replica number of a component found in the sample falls below t, the component is identified as endangered. We define the false negative probability for the lth component, denoted as P^l_{neg}, to be the probability of finding j ($j > t$) replicas from the sample; and the false positive probability for the hth component, denoted as P^h_{pos}, to be the probability of finding j ($j < t$) replicas from the sample. These are given below, where $K = \omega k d$ represents the total number of sampled tenants.

$$P^l_{neg} = \sum_{j=t}^{j=\infty} P_r(j) = 1 - \sum_{j=0}^{j=t-1} \binom{K}{j} (r_l)^j (1 - r_l)^{K-j} \quad (1)$$

$$P^h_{pos} = \sum_{j=0}^{j=t-1} P_r(j) = \sum_{j=0}^{j=t-1} \binom{K}{j} (r_h)^j (1 - r_h)^{K-j} \quad (2)$$

From the above, given the desired false positive rate for non-endangered components, and the desired false negative rate for endangered components, we can determine the appropriate value for random walk sample size. Given $\omega \cdot k$, maximizing the number of walkers k produces lower delay than maximizing ω. Thus, in practice, we may want to set k to be the maximum of component managers' node degree (i.e. number of component manager neighbors).

Finally, the initiating component manager computes the aggregated result from all local records it receives from other component managers and generates a new system record. Combined with the system records in the walkers, the component manager produces an up-to-date estimation of component population by choosing the maximum value from these records. If the estimated value is still below the threshold, the component is confirmed as an endangered component.

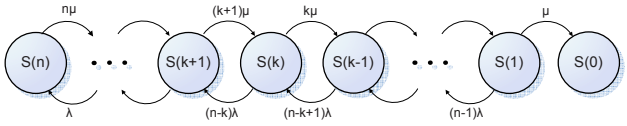


Figure 4: Markov chain describing component lifetime

3.3 Analytical Study

We now present an analytical study to quantify the benefit of our selective dynamic replication scheme. Recall that our system performs dynamic replication when the estimated number of replicas falls below a certain threshold. The metric of interest here is how many replicas (or popularity level r_t) are needed to achieve a balance between a long component lifetime and low replication cost. We employ a Markov model [19] to analyze the number of replicas that can enable a system to achieve a specified expected lifetime.

We analyze how many component replicas are needed for endangered components to achieve a long lifetime in the following way. Each state S in the Markov chain represents the number of current replicas of the component c_i in the system. Thus, the lifetime of a component is the average time starting from an initial state $S(n)$ until reaching state $S(0)$. There are two major factors affecting state transitions, the component replica unavailable rate μ, and the replica generation rate λ. Component replica unavailability can be caused by node departure or component deletion. We define r_r as the replica deletion degree, representing the probability for a tenant to leave the system or delete the component c_i. For endangered components, replica generation is mainly attributed to proactive replication. Thus, the expected lifetime t_s is denoted as follows, where $\gamma = \lambda/\mu$, with γ representing the system capability of compensating for a replica which has left.

$$t_s = \frac{1}{\mu} P_n(\gamma) = \frac{1}{n\mu} \sum_{k=0}^{k=n-1} \sum_{j=0}^{j=k} \frac{\binom{n}{j}}{\binom{n-1}{k}} \gamma^{\,k-j} \tag{3}$$

Our result leads to a similar conclusion with the paper [19]. The result shows a long component lifetime can be achieved by either maximizing n through maintaining a large amount of replicas, or by maximizing γ through aggressive repair. For example, if we set μ to 1, a mean lifetime of 10^{10} can be either achieved with γ equaling to 1 and n being above 40, or γ equaling to 10 and n being more than 12. We believe aggressive replication when the number of replicas falls below the replication threshold results in better system performance than having a large number of replicas for any component. γ is affected by both node unavailability degree and replication rate. The latter can be adjusted by replicating with multiple copies in compensation for one leaving replica. We call this number the repair ratio, denoted as R. The proposed replication strategy is to keep a relatively low replication threshold and set a moderate repair ratio to guarantee a long lifetime, even under high component replica unavailable rate.

4. EXPERIMENTAL EVALUATION

In this section, we first present system implementation and experiment setup. We then describe our experimental results, and lessons learned from the experiments.

4.1 System Implementation and Experiment Setup

We have implemented a prototype of the highly available component sharing system in C++. The component manager algorithm is fully implemented. Only the message delivery between different nodes is simulated, in order to perform large-scale experiments. We have implemented all dynamic tenant operations in the simulator, such as bootstrap, join, leave, ping, pong, query, queryhit, etc. Each tenant, when joining the system, needs to first contact the bootstrap server to get a random list of component managers with which to connect. Connections are established through three-way handshakes.

Our system emulates dynamic tenants in a cloud system in the following way. Tenants dynamically switch among three states: offline, idle, and active. To become active, an offline tenant must first enter the idle state. The transition from the idle to the active state is accompanied by the issuance of a component query. After a successful query, a tenant in the online state has a probability u_r (node churn rate) of going offline, and a probability $1 - u_r$ of instead changing to the idle state. Tenants in the idle or active states can reply to queries and remain active in the system. This model follows the design concept of [27]. Key system parameters were chosen based on measurement studies [7] [22] of real world P2P systems that resemble autonomous multi-tenant cloud systems.

We deploy 2,500 tenants and 250 component managers in the component sharing system. The ratio between the number of component managers and the number of tenants is heuristic. If the ratio of component managers is too low, component managers may be overloaded since they have to interact with a number of tenants. However, a high ratio of component managers may result in inefficiency in random walk sampling since random walkers of given length may only be able to collect very limited tenant information. There are a total of 5,000 unique components and over 100,000 component replicas produced during each experiment run. Component queries follow the Zipf distribution with a zipf-exponent of 1.0. This has previously been found to be adequate to capture the skew in file queries for P2P file-sharing systems [11], and was used also in the related work such as [24] and [28]. Initial component distribution also follows a quasi-zipf distribution according to [29], and we use an exponent of 0.7 as in previous works. Mean component size can be adjusted from 128MB to 2GB. The network topology follows the transit-stub model. A component manager can maintain up to 10 connections to other component managers and also up to 10 connections to cloud tenants.

We evaluate our scheme using three sets of important metrics: The first set of metrics is *availability prediction accuracy*, in terms of the true positive rate A_{tp}, and the false positive rate A_{fp}. The rates are calculated as follows,

$$A_{tp} = \frac{N_{tp}}{N_{tp} + N_{fn}} \tag{4}$$

$$A_{fp} = \frac{N_{fp}}{N_{fp} + N_{tn}} \tag{5}$$

where N_{tp} denotes the number of true endangered components that get correctly identified as endangered; N_{tn} denotes the number of non-endangered components that are correctly identified as non-endangered; N_{fp} denotes the number of non-endangered components that are incorrectly identified as endangered; and N_{fn} denotes the number of true endangered components that are incorrectly identified as non-endangered. We obtain these values by comparing the prediction results with the ground truth. The second set of metrics is *availability*, which is the fraction of components that maintain at least one copy in the system at all times. Thus, it is measured as the number of components that always have one or more copies, divided by the total number of components. The third set of metrics is *storage cost*, which is measured as the actual storage used for saving extra copies of all unique components in the system.

For comparison, we have also implemented four other alternative schemes: the no replication scheme, the full replication scheme,

the queue monitoring replication scheme, and the local monitoring replication scheme. The no replication scheme represents a system that has no availability management, where the dynamics of tenants may easily cause permanent loss of some components that have few replicas. The full replication scheme keeps track of all unique components and keeps an extra copy for all unique components. The queue monitoring replication scheme refers to a distributed availability management scheme similar to the first part of our scheme. Component managers will label a small fraction of the least-frequently queried components as endangered. This differs from our scheme in that it does not use random walk sampling to further identify endangered components. All components locally labeled as endangered will get replicated on the component managers. Finally, in the local monitoring replication scheme, component managers also maintain a list of least-frequently queried components. However, this list is only updated for queries that are from the tenants that directly connect with the component manager. In other words, the local monitoring replication scheme only monitors local queries. Similar to the queue monitoring replication scheme, component managers label a small fraction of the least-frequently queried components as endangered, and replicate all these components on the component managers.

We evaluate the proposed scheme in four aspects. First, the accuracy of identifying endangered components is evaluated. Second, the ability to provide availability management (i.e., successfully maintain sufficient copies of all components) in the presence of various rates of component deletion, is investigated. Third, the proposed scheme is compared with other schemes with respect to the additional storage cost. Lastly, the scheme overhead is evaluated.

4.2 Results and Analysis

The first experiment investigated the accuracy of our scheme in identifying endangered components. Figure 5 compares our scheme with the queue monitoring scheme, and the local monitoring scheme. The comparison is in terms of the true positive rates and the false positive rates, as a function of the fraction of the local LFQ queue labeled as local endangered components. This fraction ranges from 5% to 30%. In this experiment, the random walk sampling threshold is 2, which means a component is identified as non-endangered when our scheme finds more than two copies of it from random walk sampling. Note that the threshold is a heuristic parameter. If the ratio of the number of component managers to the number of tenants is low, random walk sampling can locate more component replicas resulting in higher estimation accuracy. On the contrary, if the ratio of component managers is high, a lower threshold is needed since less replicas can be located through random walk sampling. The node churn rate is 0.1, which means that a node has 0.1 probability of going offline after a successful query. By comparing the identified results with the ground truth, we calculate N_{tp}, N_{tn}, N_{fp} and N_{fn}. Thus, we can calculate the true positive rate A_{tp} and false positive rate A_{fp}.

Figure 5 shows that our scheme can achieve much higher or similar true positive rates than other schemes but with much lower false positive rates. As expected, the true positive rate increases when the fraction of the LFQ queue that is marked as endangered is increased. This is because with a higher probability of labeling, our scheme may have more chances to detect endangered components in the system. At the same time, the false positive rate increases with the increasing of this probability, since there is also more chance of including non-endangered components. Note that labeling a smaller fraction only causes a small reduction in the true positive rate, but can reduce the false positive rate greatly. As expected, our scheme achieves a similar true positive rate and a lower false positive rate, compared with the queue monitoring scheme. This is because our scheme further eliminates non-endangered components after obtaining the queue monitoring results. For the local

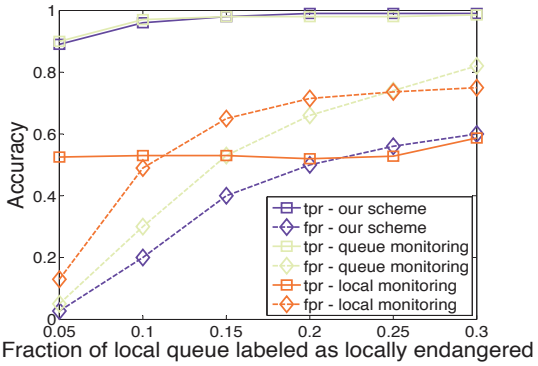


Figure 5: Accuracy in identifying endangered components.

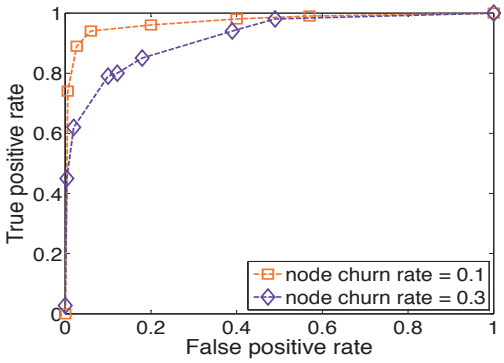


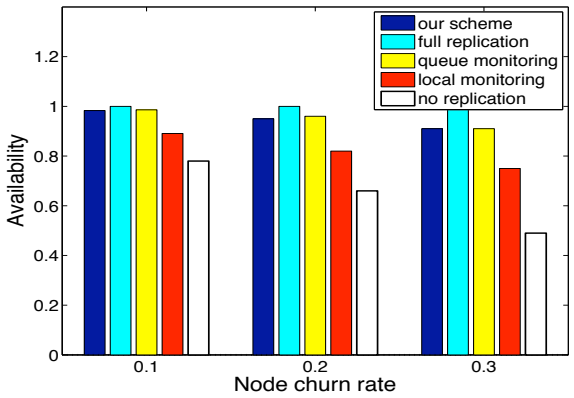
Figure 6: ROC curves for identifying endangered components.

monitoring scheme, the true positive rate is low and the false positive rate is relatively high. This is because component managers cannot accurately capture the endangered components due to the limited query pattern observations.

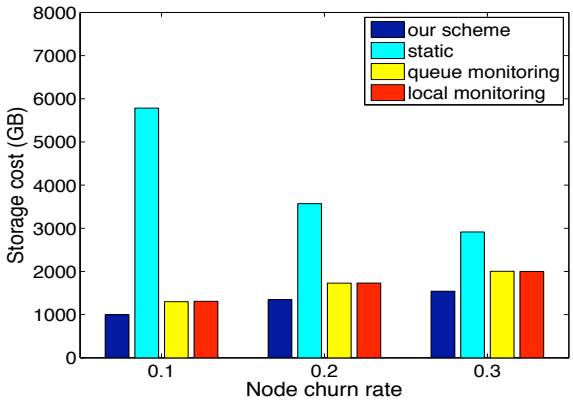
We also evaluate the performance of our scheme using a Receiver Operating Characteristic (ROC) curve. To generate the ROC curve, we adjust the fraction of the local LFQ queue for labeling local endangered components. That is, component managers label the last 1%, 3%, 5% through 20% of the LFQ queue as local endangered components. Our scheme can then identify a set of components as endangered components through both local and system-wide investigation. Results were collected from multiple runs. We generate two curves with node churn rate u_r equaling to 0.1 and 0.3 respectively. Figure 6 shows the ROC curves, with the X-axis being the false positive rate. We observe our scheme can achieve good accuracy with ROC curves close to the upper left-hand corner in the diagram, even under a high node churn rate.

We now evaluate the availability maintenance ability of our scheme and compare it with four other schemes. We also show the corresponding storage cost for better comparison. Figure 7(a) compares our scheme with the other four schemes in terms of availability, by varying the node churn rate from 0.1 to 0.3. As expected, when there is no availability management scheme, the more quickly tenants go offline, the more components become unavailable. The system may lose valuable resources permanently. The full replication scheme achieves the best availability since it keeps a copy of all components all the times. Our scheme and the queue monitoring replication scheme achieve similar performance but with different storage cost, which will be shown next. The local monitoring replication scheme cannot effectively provide availability management due to the incapability of capturing endangered components. We

learn that monitoring queries from a small range of tenants does not help predict system availability.



(a) Availability comparison.



(b) Storage cost comparison under different node churn rate.

Figure 7: Availability and storage cost comparisons of different schemes.

Figure 7(b) compares our scheme with the full replication scheme, the queue monitoring scheme and the local monitoring scheme in terms of storage cost, while varying node churn rate from 0.1 to 0.3. We observe that our scheme consistently incurs the lowest storage cost. The queue monitoring replication scheme and the local monitoring replication scheme have similar storage cost because they both replicate a certain fraction of components in the monitoring queue. Both schemes performs worse than our scheme because different component managers independently replicate the same endangered component. However, our scheme can alleviate the problem by gaining a global view from the system-wide random walk availability investigation. With increasing system dynamics, our scheme enjoys an increasing advantage over the queue monitoring scheme and the local monitoring scheme. Clearly, the full replication solution has the largest storage cost because each component manager has to replicate all unique components from each tenant with which it connects.

Figure 8 compares the storage cost of our scheme with the other three availability management schemes, where mean component size ranges from 128MB to 2GB. The node churn rate in this experiment is $u_r = 0.1$. Results indicate that our scheme shows improved storage savings, with larger component sizes.

Compared with other availability management schemes, the overhead of our scheme mainly comes from random walk investigation. We evaluate the overhead by measuring the number of random walk messages, for different network sizes ranging from 500 to 2,500 nodes. The number of component managers is about 10% of nodes under each scenario. Figure 9 shows the total number of random walk messages during an experiment. The number of random walk messages increase almost linearly with network size. This is because the number of component managers that can issue random walks increases linearly with network size. Note that the number of random walks increases with the increasing of node churn rate. This is mainly because the more dynamic the system is, the more components may be endangered. Thus, more random walks are triggered to investigate component availability. Also, we measured the average number of random walks that a component manager issues per second as being less than 0.002. Compared with component query frequencies, which in general has an upper bound of 8 queries per node per second, the random walk investigation introduces negligible system overhead.

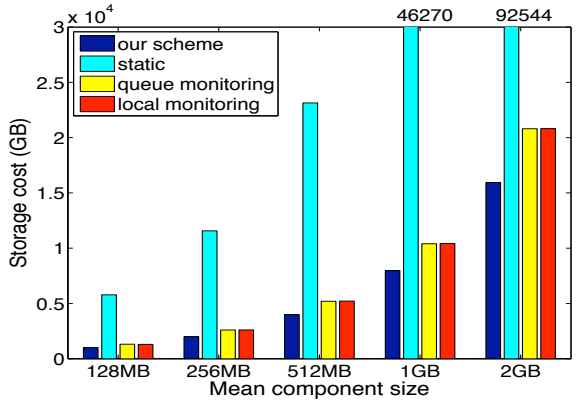


Figure 8: Storage cost comparison of different schemes under different mean component size.

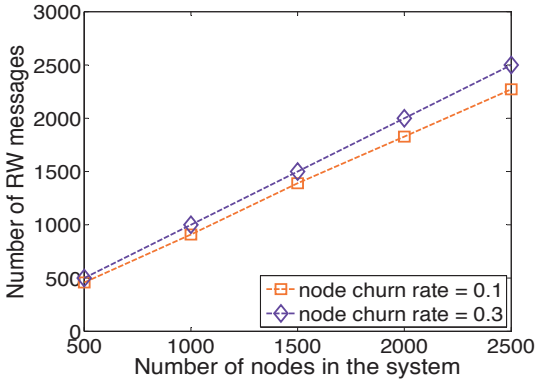


Figure 9: System overhead in terms of random walk messages.

5. RELATED WORK

The availability problem has attracted considerable attention [5, 20,26] in dynamic decentralized systems such as P2P systems. P2P Storage systems that are based on structured P2P networks, such as CFS [9], PAST [21], OceanStore [14] and pStore [4] achieve availability through exact-copy replication or erasure coding. For example, CFS replicates each file block on k CFS servers, which are immediately after the block's successor on the Chord ring. PAST replicates files in a similar way; however, it stores whole files rather than blocks. The availability of a file is determined by the node

availability of the k nodes. In OceanStore, an object is erasure-coded into many fragments and distributed among the servers. A file may be reconstructed as long as enough fragments are retrievable. The pStore uses exact-copy chunk replication and file owners are responsible to send replicas to target nodes. However, their assumptions are that indexes are always available and correct, and nodes always fulfill their responsibilities and store what they are assigned. In large-scale multi-tenant shared computing systems, it is hard if not impossible to enforce a uniform structure over different content providers.

Replication strategies proposed for unstructured P2P networks usually aim at optimizing the search performance, instead of maintaining availability for all files, and may make under-provisioned objects even rarer. Cohen and Shenker discussed various replication strategies for unstructured networks in [8], including uniform, proportional and square-root replications. The uniform strategy is to replicate everything equally, which is far from optimal in terms of performance. The proportional strategy is to make the number of replicas be proportional to the objects' query rates. The square-root strategy lies between the above two, and is shown to minimize the expected search time for successful queries. Tewari and Kleinrock showed the proportional replication has many per-node advantages and can be achieved in a distributed manner using LRU local storage management algorithms [23, 24].

Some other research work proposes hybrid P2P networks, which constructs a DHT on top of an unstructured network, such as PeerStore [15]. In a dynamic environment where there is a high degree of churn, not only files but also the DHT structures should be replicated to ensure availability. The underlying complexity and robustness problems seem to be the most arduous problems. Cheng and Joung [6] used Bloom filters to index files and gossip among the peers in order to better locate rare files. However, their work aimed at improving searching performance rather than availability, and there was no scheme to identify rare files and perform replication.

Previous work on content distribution networks mostly focuses on efficiently and reliably serving users' requests. CoDeeN employs heartbeat messages to monitor node health and avoids problematic proxies when directing users' requests [25]. LOCKSS is a peer-to-peer digital library system targeted to achieve long-term digital preservation [17]. Each peer in the system periodically launch opinion polls over its content and tries to maintain a fresh and valid copy. Different from our work, both systems assume that data objects are always available and retrievable.

Other work related to this paper investigated methods for in-network aggregation in distributed systems. Li and Lee exploited an in-network filtering technique "netFilter" to identify frequent items in [16]. It is based on hierarchical aggregation and relies on constant maintenance of the hierarchy to compute precise global values of frequent items. Chu et al. presented a framework of synopsis diffusion computing approximate global state in sensor networks [18]. Their method uses synopses designed to avoid double-counting, thus enables the use of arbitrary multi-path routing schemes in energy-sensitive sensor networks. Both work did not address availability issues. In addition, in our scheme, it is not necessary to compute precise global values.

6. CONCLUSION

In this paper, we have presented a novel highly available component sharing system for large-scale multi-tenant cloud systems. The system employs dynamic predictive replication to maintain high availability of all components with low extra storage cost. Instead of replicating all components uniformly, our system tries to discover those endangered components and replicate those subset of components only. We perform component availability prediction using a hybrid scheme: 1) leverage query results to remove local popular components from the "endangered" list; and 2) employ a random walk sampling scheme to further conduct system-wide component popularity investigation. The system then performs proactive replication to preserve those endangered components. We have implemented a prototype of the proposed component sharing system. Our experimental results show that our scheme is both efficient and light-weight. Particularly, we observe that our predictive replication scheme can 1) achieve up to 99% availability with about 15% of the full replication cost; and 2) provide better or similar availability than other selective replication schemes but with much less storage cost.

7. ACKNOWLEDGMENT

This work was sponsored in part by NSF CNS-09-1-5861, NSF CNS-09-1-5567, U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI), IBM Faculty Award, and Google Research Award. Any opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF, ARO, or U.S. Government.

8. REFERENCES

[1] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[2] Microsoft Azure Services Platform. http://www.microsoft.com/azure/default.mspx.

[3] The gnutella protocol specification 0.6, 2002. http://rfc-gnutella.sourceforge.net.

[4] Christopher Batten, Kenneth Barr, Arvind Saraf, and Stanley Trepetin. pStore: A secure peer-to-peer backup system. Technical Report Technical Memo 632, MIT Laboratory for Computer Science, October 2001.

[5] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, February 2003.

[6] An-Hsun Cheng and Yuh-Jzer Joung. Probabilistic file indexing and searching in unstructured peer-to-peer networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 50(1):106–127, 2006.

[7] Jacky Chu, Kevin Labonte, and Brian Neil Levine. Availability and popularity measurements of peer-to-peer file systems. Technical Report 04-36, Deptartment of Computer Science, University of Massachusetts, June 2004.

[8] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of SIGCOMM '02*, Pittsburgh, PA, August 2002.

[9] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001.

[10] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *Proceedings of the 23th IEEE International Conference on Computer Communications (INFOCOM '04)*, Hong Kong, China, March 2004.

[11] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, NY, USA, October 2003.

[12] Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of ACM SIGOPS Operating Systems Review (SOSP)*, pages 205–220, 2007.

[13] Kate Keahey and Tim Freeman. Science clouds: Early experiences in cloud computing for scientific applications. In *Proceedings of Cloud Computing and Its Applications (CCA)*, Chicago, IL, October 2008.

[14] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Hakim Weatherspoon Sean Rhea, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*, November 2000.

[15] Martin Landers, Han Zhang, and Kian-Lee Tan. Peerstore: Better performance by relaxing in peer-to-peer backup. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P '04)*, Zurich, Switzerland, August 2004.

[16] Mei Li and Wang-Chien Lee. Identifying frequent items in p2p systems. In *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS '08)*, Beijing, China, June 2008.

[17] Petros Maniatis, David S. H. Rosenthal, Mema Roussopoulos, Mary Baker, TJ Giuli, and Yanto Muliadi. Preserving peer replicas by rate-limited sampled voting. 37(5):44–59, 2003.

[18] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of International Conference on Embedded Networked Sensor Systems (SenSys '04)*, Baltimore, MD, USA, November 2004.

[19] Sriram Ramabhadran and Joseph Pasquale. Analysis of long-running replicated systems. In *25th IEEE International Conference on Computer Communications (INFOCOM '06)*, Barcelona, Catalunya, Spain, April 2006.

[20] Rodrigo Rodrigues and Barbara Liskov. High availability in dhts: Erasure coding vs. replication. In *Proceedings of the 4th International Workshopon Peer-to-Peer Systems IPTPS'05*, February 2005.

[21] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Banff, Canada, October 2001.

[22] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[23] Saurabh Tewari and Leonard Kleinrock. Analysis of search and replication in unstructured peer-to-peer networks. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS '05)*, 2005.

[24] Saurabh Tewari and Leonard Kleinrock. Proportional replication in peer-to-peer networks. In *25th IEEE International Conference on Computer Communications (INFOCOM '06)*, Barcelona, Catalunya, Spain, April 2006.

[25] Limin Wang, KyoungSoo Park, Ruoming Pang, Vivek Pai, and Larry Peterson. Reliability and security in the codeen content distribution network. In *Proceedings of USENIX 2004 Annual Technical Conference*, Boston, MA, USA, June 2004.

[26] Hakim Weatherspoon and John Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of the 1st International Workshopon Peer-to-Peer Systems IPTPS'02*, March 2002.

[27] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE '03)*, Bangalore, India, March 2003.

[28] Matei Zaharia and Srinivasan Keshav. Gossip-based search selection in hybrid peer-to-peer networks. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS '06)*, Santa Barbara, CA, February 2006.

[29] Shanyu Zhao, Daniel Stutzbach, and Reza Rejaie. Characterizing files in the modern gnutella network: A measurement study. In *Proceedings of the 13th Annual Multimedia Computing and Networking (MMCN '06)*, San Jose, California, January 2006.