2012 41st International Conference on Parallel Processing

Provision of Storage QoS in Distributed File Systems for Clouds

Chien-Min Wang
Institute of Information Science
Academia Sinica
Taipei, Taiwan
cmwang@iis.sinica.edu.tw

Tse-Chen Yeh
Institute of Information Science
Academia Sinica
Taipei, Taiwan
tcyeh@iis.sinica.edu.tw

Guo-Fu Tseng
Research Center for Information Technology Innovation
Academia Sinica
Taipei, Taiwan
cooldavid@citi.sinica.edu.tw

Abstract—**In this paper, we present a distributed file system with storage QoS provision for cloud computing. The objective is to maximize the aggregated disk bandwidth utilization with the assurance of allocated disk bandwidth for each data transfer. First, we implement a distributed file system based on the ECNP model, which avoids matchmaker overloading and the lack of up-to-date information in the original matchmaking model. Next, resource selection policies for resource management are investigated to improve bandwidth utilization. Furthermore, we propose several dynamic replication strategies to balance the workload of resource providers. Experiments are conducted in real systems to study the performance on storage QoS with different resource selection policies and dynamic replication strategies.**

Keywords- Cloud Computing; Distributed File Systems; Dynamic Replication; Resource Management; Storage QoS

I. INTRODUCTION

Following the rapid growth of cloud computing, increasing numbers of data-intensive applications with real-time requirements are been deployed in large-scale distributed storage environments. A very large amount of real-time applications, e.g., video streaming, virtual reality, scientific data gathering, digital libraries, need Quality of Service (QoS) support for data access [10], and the side effect of failing to meet such requirements will be drastically noticed when the system scale is enlarged. Furthermore, popular high-throughput data-intensive processing, such as MapReduce, also needs the QoS to guarantee the stable gathering of throughput for concurrent data access [16]. Although several distributed file systems (DFS) for cloud computing, including Hadoop DFS (HDFS) [23] and the Google File System (GFS) [12], have been proposed, the storage QoS support for DFS is still under development. In addition, most well-known techniques to cope with QoS support in distributed file systems principally focus on the flow control and the network management among heterogeneous resource clusters due to the file system's inherence. Certain storage QoS mechanisms based on clusters or grid systems adopt parallel data retrieval and static replication to achieve the primary QoS, but the lack of data migration will make it difficult for the QoS provision to be assured in a long-running practical system. Therefore, how to provide storage QoS in a distributed file system for cloud computing still remains in challenge.

In this paper, we aim to investigate how to provide the storage QoS which makes resource reservations and allocates required disk bandwidth to real-time applications or any data transfer that needs fixed bandwidth assurance. We will first present the design and implementation of our distributed file system, which is built on our Xen-based cloud environment with the cgroup-blkio bandwidth control mechanism. To fulfill the requirement of accessing a large amount of data, we adopt the Extended Contract Net Protocol (ECNP) model [27] to apply to our distributed file system in light of considerations of scalability and the possibility of more efficient message traffic. This protocol extends the CNP bidding-based negotiation model [24] used to manage massive storage resources by integrating a matchmaking technique to avoid excessive redundant messages.

Next, we facilitate the aggregated bandwidth utilization by using several resource selection policies to explore the possibility of allocating bandwidth from slack resource providers. Data migration techniques using dynamic replication will also be triggered in the case that all the static replicas are still insufficient to fulfill the users' requirements. Experiments are conducted in the proposed distributed file system to study the performance on storage QoS with different resource selection policies and dynamic replication strategies. The experimental results show that the combination of resource selection policies and dynamic replication strategies can provide better QoS support than without this combination of techniques. The proposed system also maximizes the aggregated disk bandwidth utilization with the assurance of allocated disk bandwidth for each data transfer.

The rest of this paper is organized as follows. Section 2 provides a review of related works on storage QoS for resource management. Section 3 briefly describes not only the system components used to build our distributed file system, but also the implementation details of our distributed file system and the Xen-based virtual environment with the bandwidth control mechanism. The resource selection policies are proposed in Section 4, and in the next section, the dynamic replication mechanisms will be investigated. The experimental results will be demonstrated in Section 6, and finally, Section 7 contains the concluding remarks.

II. Related Work

Many distributed file systems have been proposed for supporting cloud computing. HDFS [23] is one of the implementations of the GFS [12], written in Java. HDFS uses one NameNode and several DataNodes to establish the distributed file system and facilitates massive data access. Before invoking any data access operation, the user client needs to communicate with the NameNode for acquiring the Metadata, which contain the distribution information of files in the DataNodes. Kosmix proposed another GFS implementation called CloudStore [11], written in C++. It provides the similar functionalities of file operations as the HDFS does. Lustre [4] is a cluster-based distributed file system widely adopted by supercomputer systems because of its superior performance. However, the file systems mentioned above target at providing the file access service of resource management, not the storage QoS for real-time requirement.

Another category of distributed storage systems is constructed by disk arrays to provide massive virtual storage, e.g. FAB [22], Petal [14] and IceCube [31]. This type of storage system integrates several separate storage resources in order to provide a reliable, virtualized local disk storage service for users or applications. However, non-trivial design and implementation on the combination of operating system porting and certain hardware support are unavoidable. SCADS Director [25] mainly focuses on dynamic resource allocation for database application but not much effort is invested on assuring storage QoS.

Most research associated with storage QoS focus on the disk scheduler, which can be achieved by scheduling the I/O requests for disks. In order to prevent the most aggressive best-effort workload from starving the soft real-time one, several storage QoS-aware techniques have been proposed. Wu et al. [33] proposed managing the bandwidth by using a traffic shaping technique above the external disk scheduler. Bigelow et al. [9] proposed I/O reservation by using the Fahrrad disk scheduler above the Ceph file system [30]. Both of the above-mentioned techniques do not take the global resource information into account, i.e. one disk scheduler will do nothing even if the other disk scheduler overloads.

Some investigations have endeavored to provide support for storage QoS by extending the cluster file system, such as Lustre file system. Narasimhamurthy et al. [18] exploited both the bandwidth throttling and bandwidth reservation to provide storage QoS for high performance computing clouds; and Mu et al. [17] proposed that a multi-dimensional storage QoS is guaranteed by using resource mapping and I/O command scheduling. Although CA-NFS [8] can simultaneously manage diverse resources, including network bandwidth, server I/O, server CPU, and memory utilization, most effort is concentrated on reducing latency and improving performance instead of guaranteeing QoS. All of the aforementioned techniques do not apply the concept of data replication to modern cloud computing environments.

Nikolow et al. [20] proposed the bandwidth reservation technique by using monitoring and performance prediction based on heuristic policies for virtualized heterogeneous storage resource in a cloud environment. Velusamy et al. [26] adopted data replication to offer a statistical QoS assurance based on a parallel NFS (pNFS) grid file system. Wei et al. [29] utilized QoS-aware striping with a replication model to assure availability and continuity based on multimedia storage. Wolfson et al. [32] and Ranganathan et al. [21] proposed diverse dynamic replication algorithms and strategies without considering storage QoS assurance. Although data replication has been adopted for the all of the above-mentioned distributed storage systems, the influences of data migration have been left unexplored. And even though Aqueduct [15] exploited data migration for reliability, there was little attention paid to sustained bandwidth guarantee.

III. The Proposed Distributed File System

A. System Components

The proposed distributed file system consists of the three components, of the Distributed File System Client (DFSC), Resource Manager (RM), and Metadata Manager (MM) (shown in Fig. 1), each of which was mapped one-by-one to the Requester, Storage Provider and Mapper of the ECNP model, respectively. These three components are not constrained to be deployed on the same physical machine. Due to the inherence of cloud computing, the RM can be deployed on one of the virtual machines (VM) with other VMs running certain applications on the same physical machine.

From the users' point of view, the only thing that users need to know is the IP address of DFSC, because DFSC will negotiate with RMs under the MM's management and access the requested data on behalf of users. Moreover, as a single MM is capable of being accessed by multiple DFSCs and maintains the resource list of multiple RMs, we can explore the effect of distributed storage resource with the QoS management in a multi-user environment.

The functionalities and operations of each of the system components of the proposed distributed file system are described as follows:

- The DFSC is responsible for receiving and processing the external request, sending the query of a resource list to the MM, sending Call-For-Proposal (CFP) to the RM, evaluating the bid returned from the RM by feeding the bid as input parameters of various resource selection policies, and finally launching the data access with the selected RM.
- The RM has the responsibility to register its own managed resources to the MM, to return the bid back to the DFSC after receiving the CFP, and to maintain the dynamic run-time information, e.g. the current remained storage bandwidth, of its host during the data communication.
- The MM needs to deal with two operations. The first is to collect the resource information sent from the individual RMs and maintain the global resource list. The other is to

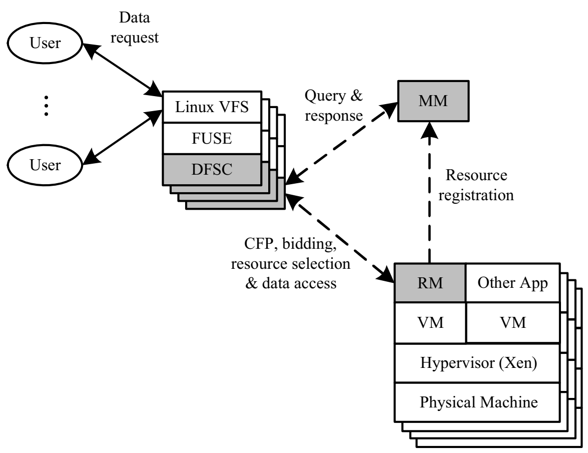


Figure 1. System components of the proposed distributed file system.

process the query from the requester and return a list of eligible RMs to the requester.

1) System Implementation

Generally speaking, the granularity of data distribution in a distributed storage system can be divided into two categories: file granularity and chunk granularity. File granularity is more intuitive than chunk granularity when implementing a distributed file system; and the chunk granularity is in widespread use within GFS and other distributed storage systems. Since our goal is to investigate the performance of storage QoS on various resource selection policies and dynamic replication strategies, we adopt the file granularity for our distributed file system in data distribution.

Traditionally, a file system is a part of the operating system. It is too difficult to develop and implement our system in the kernel space if we implement our distributed file system as an independent file system without other middleware support. Therefore, we adopt FUSE [2] for DFSC implementation to avoid most of the possible problems that occur by kernel space programming. The FUSE library assists the full functionalities of a file system as a user space program. All the operations associated with file manipulation need to be implemented as callback functions, such as `getattr`, `readdir`, `open`, `read`, `write`, `release`, `destroy` and so forth. With the assistance of the FUSE library, the user's requests received by Linux virtual file system (VFS) interface can then be delivered via the FUSE module to the DFSC without intervention from the development of any error-prone kernel modules. Moreover, the query operation for a resource list from the DFSC to the MM is implemented in the `readdir` operation and the CFP sending and resource selection algorithms are implemented in open operation. In addition, read and write operations will launch the data access with the RM determined in open operation.

Note that the global resource list maintained by the MM is the union of the resource information provided by all of the registered RMs. During resource registration, the MM also needs to maintain the integrity and consistency of the global resource list.

2) Bandwidth Control

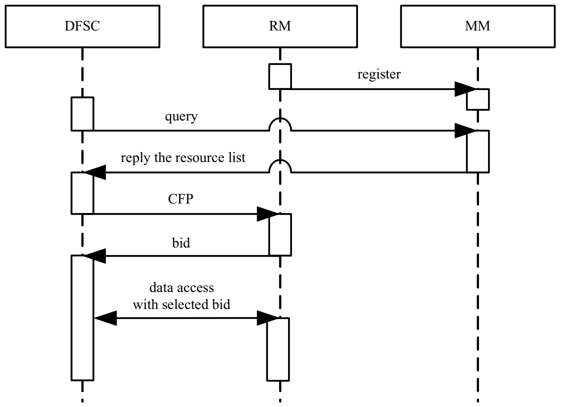


Figure 2. System implementation view of the ECNP distributed bidding model.

A great number of modern cloud computing environments are constructed using VM technology which can be categorized into three different types: para-virtualization, full-virtualization and hardware-assisted virtualization. Para-virtualization injects a new management layer, called hypervisor, between the OS and hardware. For the sake of managing the partial functionalities of the OS and I/O device drivers, the hypervisor support needs to modify the guest OS. Both Xen [7] and KVM [3] are the most well-known VMs using para-virtualization technology. Full-virtualization usually translates the binary code of the guest OS into a binary executable of the host OS by means of the so-called binary translation, e.g. VMware [6] and QEMU [5], thus requiring no modification for the guest OS as executing VM and almost no side effect will be experienced when the VM crashes. The well-known Intel VT-x [19] is the most representative of hardware-assisted virtualization. It is also notable that para-virtualization can avoid the guest OS modification if the VM is running on the VT-x supported host. In this paper, we adopt Xen to build our virtual cloud storage environment because it has the domination on the bandwidth control of block device.

With regard to the virtual cloud storage, multiple VMs may be deployed on the same physical machine using the same physical block device. This situation increases the difficulty of assuring storage QoS as the block device cannot distinguish which I/O request comes from which VM. Hence, we exploit cgroups-blkio, a subset of the control groups mechanism [1], to manage the access bandwidth of different VMs. Control groups can be used to control the behaviors of a group of tasks and their invoked child processes by means of their process ID and other parameters. For example, cgroups-blkio can manipulate or limit the block device by setting the constraints, like service time, wait time or disk sector. One subset of blkio, i.e. blkio.throttle, exploits the read_bps_device and the write_bps_device to constrain the upper bound of the disk read/write bandwidth acquired by the designated process. Consequently, we can guarantee that the sustained bandwidth of the VM respectively by isolating individual access bandwidth dispatched from the same physical block device via the cgroup-blkio mechanism.

B. *Interactions between System Components*

Specifically, our implementation has two principal differences as compared with the conceptual ECNP model: the first is the response mechanism of the bid message, and the second is the data access operation between the DFSC and the RM. The former is caused by the fact that only the DFSC can determine the selection priorities by means of various resource selection policies. Thus, in our design, every RM should in fact return the bid as the response to the CFP, instead of refusing to provide a bid. Furthermore, because the data communication between the DFSC and the selected RM is implemented by packet transmission over the network, the redundant bid rejection and bid acceptance mechanisms are eliminated.

As shown in Fig. 2, the order of initialization of our proposed distributed file system is to first invoke the MM, and then all the RMs register their own resource information in an arbitrary order after being invoked, and the DFSC is the last one to be launched to take over the whole distributed storage system. When the system initialization is completed, users can start to send the requests and then access the virtual storage system constructed by the registered RMs.

The resource management flow can be divided into three phases. (1) Resource exploration phase: in this phase, we exploit the MM to maintain the mapping between the storage capacity and the RM in the resource list registered by the RMs. Thus, the DFSC can receive the list of corresponding RMs from the MM for the next resource negotiation phase. (2) Resource negotiation phase: this phase starts from the DFSC sending the CFP to all eligible RMs. As mentioned in the previous paragraph, the DFSC's requirement will be contained in the CFP message, and the eligible RMs will respond with their own bid determined by their bidding policies to the DFSC. After that, these collected bids can be evaluated in various resource selection policies for exploring the most appropriate RM. (3) Data communication phase: in this final phase, data can be stored into the selected storage resource or be retrieved from the selected one, and storage QoS will be assured during the data transmission.

Note that the resource selection policies can benefit the storage QoS and utilization only when more than one bid is offered to the requester by the participating RMs. Hence, we need to further exploit the dynamic replication technique to prevent the case that no disk bandwidth is available in all the participating RMs. The triggering condition of dynamic replication may be a threshold value referred by the RMs. For instance, if the threshold is set too low, it may incur too many replications and degrade the efficiency of resource utilization. In contrast, if the threshold is set too high, a burst of resource requirements may lose their QoS assurance.

Moreover, if the storage system only replicates data without deleting the redundant replicas, the resource utilization will continuously downgrade. Thus, the triggering condition of data deletion is used to determine when and how the deletion operation is needed. Similarly, if the threshold is set too low, it may slacken the data deletion and degrade the efficiency of resource utilization. In contrast, if the threshold is set too high, too many operations back and forth between data replication and deletion will result in significant system overhead. Therefore, we have to investigate the reasonable thresholds and mechanisms applied to our proposed distributed file system.

IV. Resource Selection

As far as the resource selection algorithm is concerned, our goal is to achieve storage QoS by selecting the most appropriate resource from three different perspectives. The first consideration is the available bandwidth which indicates the current remaining bandwidth of storage managed by the individual RM labeled by B_{rem}.

The second factor is used to predict the possible trend of bandwidth utilization when a new request just arrives. On account of the arrival of the I/O request to the block device being unpredictable, it is hard to know in advance if requests will arrive in a sudden burst or remain idle for a long period. For this reason, we propose a historical prediction mechanism by combining the fixed accumulated count of request arrivals and the expired time instead of recording at a fixed sample rate. In order to preserve the historical record, we implement two queues in turn to accumulate the cumulative amount of bandwidth utilization and exchange if one queue reaches the specified conditions. According to the two-queue mechanism, when one queue is currently used for recording, the other queue still can be used for the historical reference to the trend prediction. Furthermore, both of the following two conditions will trigger a queue exchange: one is the sample accumulated to the specified count; the other is the queue that has exceeded the specified expired time even if the accumulated request count is not reached. No matter which condition has been triggered, the queue used to record will be changed into the other which is used for providing the historical reference, and vice versa.

Now we formally define the time period between two queues exchanged as $T_{threshold} = T_{end} - T_{start}$. As shown in Fig. 3, T_{start} is the timestamp when the request count starts to accumulate, and T_{end} is the timestamp when the aforementioned queue exchange condition is triggered. FS_{total} is the total cumulative amount of file size in which these files are being accessed during the $T_{threshold}$ period; thus $\frac{FS_{total}}{T_{threshold}}$ is the average bandwidth utilization during $T_{threshold}$, and B_{used} is the bandwidth in use when the current request arrives. Thus, $B_{used} - \frac{FS_{total}}{T_{threshold}}$ divided by two leads to a bias of the predicted trend to the median of the current bandwidth utilization and the historical bandwidth utilization. $T_{current}$ is the timestamp of the latest arriving request, so $T_{distance} = T_{current} - T_{end}$ is the distance of the timestamp between the latest arrival request and the lattermost arrival request currently used as a historical reference. $T_{distance}$ is used to judge the degree of worthwhile reference of the historical records currently used. It also implies that the larger

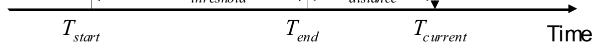


Figure 3. Timestamp parameters of historical trend prediction model.

its value, the less worthwhile the reference of the historical records. Due to the influence determined by $\frac{T_{threshold}}{T_{distance}}$, the minimum function is used as $min\left(1,\frac{T_{threshold}}{T_{distance}}\right)$ to constrain the scalable range, i.e. the evaluation result is always ≦1, for the purpose of avoiding the unexpected fluctuation of this value along with the behavior of diverse request patterns. In addition, because the calculation results may be positive or negative, which imply a positive trend or negative trend respectively, we add this factor into our equation with a plus sign, and the evaluation trend of bandwidth prediction.

The final impact factor is the bandwidth requirement for the currently requested file indicated by B_{req}. In order to estimate the influence of $B_{req}$for each RM, it will be scaled by the occupation bias ratio $\frac{T_{ocp}}{T_{ocp_{avg}}}$ belonging to the RM. The occupation time of accessing the requested file is labeled by T_{ocp}, and $T_{ocp_{avg}}$ is calculated by total occupation time divided by total number of files located on this RM, used to indicate the average occupation time of the RM in which the requested file located. In addition, because each different file has a different T_{ocp}, and different RMs have a different $T_{ocp_{avg}}$, we need to formalize the occupation bias ratio as $e^{-\frac{T_{ocp_{avg}}}{T_{ocp}}}$ scaled into a (0, 1) range.

Eventually, the three impact factors are merged as follows:

$$Bid = (\alpha \times B_{rem}) + \left(\beta \times \left(\frac{\left(B_{used} - \frac{FS_{total}}{T_{threshold}}\right) \times \min\left(1, \frac{T_{threshold}}{T_{distance}}\right)}{2}\right)\right) - \left(\gamma \times \left(e^{-\frac{T_{ocp_{avg}}}{T_{ocp}}} \times B_{req}\right)\right)$$

This equation will be used to evaluate the bid score for further resource selection, where the higher the score, the higher the selection priority. In addition, we assign the individual environment parameter along with each part, i.e. α, β and γ, and $\alpha \geqq \beta \geqq \gamma$. These environment parameters can be used to explore the optimized collocation by means of the practical experiments.

V. Dynamic Replication

Although the resource selection algorithm can benefit the resource utilization, it cannot yield a remarkable improvement when dealing with the data access hotspot. When a data access hotspot occurs, this implies that the bandwidth utilization of each RM is unbalanced, even if each datum has its replica distributed on different RMs statically, i.e. the bandwidth utilizations of some hosts are overloaded while others still have a lot of available bandwidth. In order to solve the imbalance of bandwidth utilization, a dynamic data replication mechanism is very important to the QoS performance of a distributed file system for clouds.

There are three strategies that need to be investigated regarding the dynamic replication mechanism. These strategies can take the form of questions: When should our system trigger the data replication? Which files should be replicated? And, where should we replicate files from and to?

- When to replicate: data replication should be triggered when the DFSC sends an access request to a certain RM, and the remaining bandwidth of this RM is lower than the specified threshold. The threshold is indicated by B_{TH} in percent and the RM needs to completely conform to the following conditions: (1) this RM is not currently a source endpoint of replication; (2) this RM is not currently a destination endpoint of replication; and (3) this RM has not processed data replication within 60 seconds.
- What to replicate: when data replication has been triggered, we choose the busiest files to be replicated. The degree of business is judged by the frequency of the file being requested and the first N_{BF} file that can be dynamically replicated.
- Where to replicate: we detail the operations that should be taken in source and destination endpoints respectively when processing the data replication:
  (1) Source endpoint: in accordance with the first N_{BF} files mentioned above, the RM will query the MM to reply with a resource list that contains the list of RMs without any replica of the files that need to be replicated. Afterward, we will choose N_{REP} RMs to randomly replicate the data according to the aforementioned resource list. We assume N_{CUR} is number of replicas of a file, and N_{MAXR} is the upper bound of the number of replicas in our system. Therefore, if $N_{REP}+N_{CUR}>N_{MAXR}$, then N_{REP} should be modified to the value $N_{MAXR}-(N_{CUR}-1)$. In other words, dynamic data replication will at the very least be processed one time; and if the replication exceeds the upper bound of the number of replicas, the RM will delete the replica that exists on itself. In addition, each RM should reserve B_{REV} as the available bandwidth for transferring the replicated data, and the RM will be selected as source only when $B_{REV} \geqq K \times$ Bandwidth of the designated file.
  (2) Destination endpoint: Since the source endpoint will randomly select the destination endpoint for data replication, we permit the destination endpoint to reject the replication request if any of the following conditions is matched: (1) the destination RM already has the requested replica; (2) the remaining bandwidth of the destination RM is lower than B_{REV}. If this condition is allowed, it may incur nested-replication; and (3) the remaining bandwidth of the destination RM is lower than B_{TH}.

With the integration of the dynamic replication mechanism and the proposed distributed file system, the constructed storage management system can resist the hotspot data access and further assure the storage QoS by balancing the bandwidth utilization.

VI. Experimental Results

Because our goal is to build a distributed file system for clouds with support of storage QoS, we need a benchmark of disk access that is traced with storage QoS constraints. Nevertheless, to the best of our knowledge, most of the released disk traces are gathered from mail servers, web servers, source code servers and so forth. In addition, video traces are only limited by playing a single video but not the accessing of trace of media files on a distributed file system. For this reason, we conduct experiments on a real system instead of traces. We select 1,000 video files with different bit rates and popularity ratings that were extracted from YouTube, replicate each of them as three replicas and then distribute these three replicas randomly into 16 RMs. Furthermore, the input access pattern is generated by choosing files randomly with a probability derived from the file popularity. In other words, this input pattern can guarantee that the files with higher popularity will be accessed more times in a fixed time interval. The timestamp associated with the file access is calculated by a negative exponential distribution (NET) [13]. Thus, the equation of the request arrival time is $f(x) = -\beta lnU$. In this equation, $U(0,1)$ is a random distribution function with the value ranging from 0 to 1, and β is the cumulative mean arrival time.

A. Virtual Cloud Environment Settings

Our virtual cloud environment consists of 25 Xen-based VMs, i.e. 16 RMs, 1 MM and 8 DFSC, distributed on 5 physical machines, each of which has two Intel E5620 2.4GHz 4-core 8-thread processors, 72GB memory, and a 1TB local disk, which can yield a total of 128Mbps, i.e. 16MB/s, of sustained disk bandwidth to be dispatched to VMs located on the local disk. Although we use a single MM in the experiments, a distributed MM can be achieved by a Distributed Hash Table (DHT) as shown in [28]. Each VM used for the RM is virtualized as a host with an Intel E5620 2.4GHz 1-thread processor, 1GB memory and 16GB disk. Because real distributed storages in cloud environments might be heterogeneous in storage capability, we adopt an imbalance strategy for resource deployment by giving two extra large RMs with 128Mbps of bandwidth, i.e. RM1 and RM9; four RMs with 19Mbps, i.e. RM2, RM3, RM10 and RM11; and the rest of the RMs with 18Mbps.

As mentioned in section 3.2.2, the bandwidth of an individual VM acquired from the physical local disk is controlled by the cgroup-blkio mechanism. In practice, a Xen-based VM has its corresponding loop kernel thread, and the virtual storage in conjunction with the VM is mapped to the corresponding loopback device. Hence, the bandwidth isolation between individual VMs on the same physical disk can be achieved by separately joining the loop kernel thread process ID of the associated VM and the loopback device number of the virtual storage into the blkio.throttle.read_bps_device group.

For the purpose of simulating multi-users access patterns, the request scheduler will send the request according to the request arrival timestamp recorded in the generated access pattern to the corresponding DFSC, respectively. Moreover, the request scheduler can designate the startup time of the simulation so that the multi-users access pattern can be guaranteed to be launched simultaneously. All of the experiments are conducted by using the access pattern of 256 users unless explicitly specified otherwise.

1) Bandwidth Allocation Scenario

To gather sufficient statistics, the total simulation time of each configuration is 2 hours, and the mean value of the request arrival time is 300 seconds in the access pattern. Furthermore, due to the absence of a definitive metric for measuring the storage QoS, we investigate the effect of our resource selection policies and dynamic replication strategies in two different bandwidth allocation scenarios, i.e. firm real-time and soft real-time. Firm real-time allocation indicates that the requested file will fail to open when none of the RMs can provide sufficient bandwidth to serve the request. That is, no bandwidth will be allocated to the access request if the file open failed. Thus, the fail rate of opened files is the criterion when the firm real-time allocation scenario is adopted.

On the other hand, soft real-time allocation means that whether or not the maximum bandwidth is exceeded, the bandwidth will always be allocated if requested. This scenario uses the over-allocate ratio $R_{OA} = \frac{S_{OA}}{S_{TA}}$ as the criterion for different configurations, where S_{OA} is the total bytes that exceeds the maximum accessible bandwidth and S_{TA} is the total bytes assigned to this RM (shown in Fig. 4).

B. Resource Selection

We investigate the resource selection algorithms by configuring different collocations of the impact factor (α, β, γ) pair, as mentioned in section 4. For instance, policy (1,0,1) indicates this configuration only takes α and γ into account, and the factor β with 0 weight will be ignored in this simulation. In addition, policy (0,0,0) is the resource selection by choosing the RM randomly without any selection policy being involved.

1) Soft Real-time Allocation

To explore the influence of impact factor (α, β, γ) pair with soft real-time allocation scenario, we first investigate the over-allocate ratios among the various access patterns with different numbers of users as shown in Table I. The rapid increase of the over-allocate ratio along with the increase of the number of users indicates that some RMs exceed their affordable bandwidth in soft real-time allocation scenario. Compared with random selection, the experimental results show that policy (1,0,0) performs much better than policy (0,0,0) for the over-allocate ratio. The other selection policies also do not show a noticeable improvement over policy (1,0,0). Detailed information of the over-allocate ratios for each RM for 256 users is shown in Table II. It can be observed that no matter which policy is used, some RMs may be overloaded whereas other RMs still have available bandwidth.

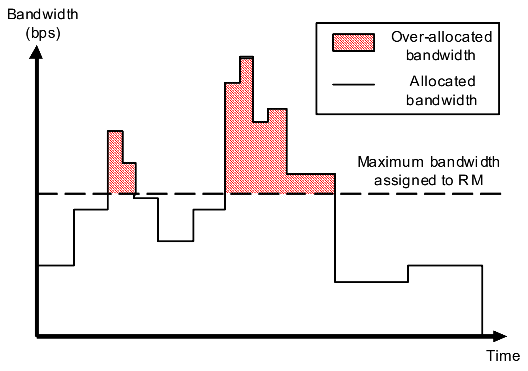


Figure 4. The over-allocate situation of soft real-time scenario. The dash line indicates the maximum bandwidth assigned to the RM.

2) *Firm Real-time Allocation*

The firm real-time scenario adopts the same environment settings with the exception of the allocation scenario. The experimental results are shown in Table III. Similar to the soft real-time scenario, the fail rate is greatly improved by using policy (1,0,0) as compared with policy (0,0,0). Furthermore, policy (1,0,1) has a better performance for 128 users and policy (1,1,0) has a better performance for 192 and 256 users. However, the performance differences are all lower than 1% as compared to policy (1,0,0).

In order to compare the bandwidth utilization, the aggregated bandwidths of the two extra large RMs and of the rest RMs are shown in Fig. 5. Though RMs with a smaller bandwidth exhausted almost all available bandwidth in both policies, policy (1,0,0) can squeeze more bandwidths than policy (0,0,0) in the two extra large RMs by selecting replicas existing in these RMs. Note that policy (1, 0, 0) utilizes at most 12MB/s of the two extra large RMs, but the maximum available bandwidth of these two RMs is 32MB/s. This indicates that selection policies on static replication have their limitations.

C. *Static vs. Dynamic Replication*

As shown in Table II and Fig. 4, selection policies on static replication have their limitations. To fully utilize the available bandwidths of all the RMs, it is necessary to adopt the dynamic replication mechanism. Several parameters can be fixed for the following experiments in order to simplify the exploration on dynamic replication: the B_{TH} used to trigger the replication operations is set to 20% of the total dispatched bandwidth of each RM; replication speed is set to 1.8Mbit/sec; the first N_{BF} files is set to cover 50% of the total access count; and the B_{REV} reversed for destination RM is set to 2 × Bandwidth of designated file. We shall use Rep(N_{REP}, N_{MAXR}) to indicate the replication strategy that replicates N_{REP} copies at a time and the maximum number of copies is N_{MAXR}. For performance comparison, experiments are conducted using the following four strategies.

(1) Static replication: distribute 3 replicas statically without triggering dynamic replication.

(2) Baseline strategy: replicate 3 copies at a time and the maximum number of copies is 8.

(3) Rep(1,8) strategy: replicate 1 copy at a time and the maximum number of copies is 8.

(4) Rep(1,3) strategy: replicate 1 copy at a time and the maximum number of copies is 3.

TABLE I. OVER-ALLOCATE RATIO IN SOFT REAL-TIME ALLOCATION

# of users / (α, β, γ)	64	128	192	256
(0, 0, 0)	1.447%	6.539%	16.325%	24.595%
(1, 0, 0)	0.000%	0.059%	2.070%	9.771%
(1, 0, 1)	0.000%	0.043%	2.102%	9.793%
(1, 1, 0)	0.000%	0.062%	2.281%	9.543%
(1, 1, 1)	0.000%	0.063%	2.215%	10.007%

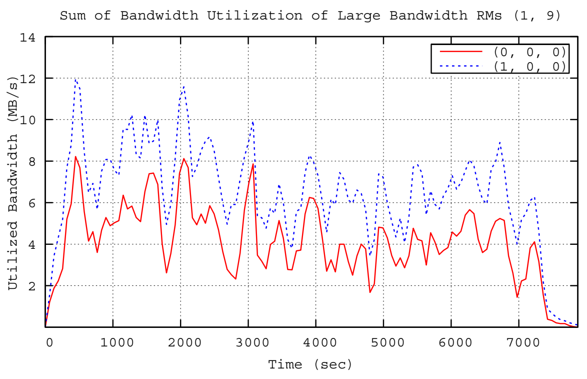
TABLE III. FAIL RATE ON AVERAGE IN FIRM REAL-TIME ALLOCATION

# of users / (α, β, γ)	64	128	192	256
(0, 0, 0)	0.070%	1.344%	7.028%	15.525%
(1, 0, 0)	0.000%	0.448%	3.825%	11.087%
(1, 0, 1)	0.000%	0.310%	4.065%	11.236%
(1, 1, 0)	0.000%	0.483%	3.604%	11.005%
(1, 1, 1)	0.000%	0.345%	4.045%	11.038%

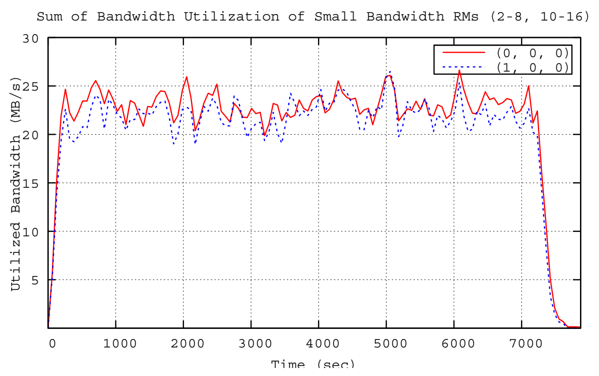
TABLE II. OVER-ALLOCATE RATIO OF EACH RM IN SOFT REAL-TIME ALLOCATION WITH 256 USERS.;THE ASTERISK SIGN INDICATES THE RM WITH EXTRA LARGE BANDWIDTH (16MB/S)

RM# / (α, β, γ)	1(*)	2	3	4	5	6	7	8
(0, 0, 0)	0.000%	46.385%	0.009%	16.712%	18.089%	17.819%	7.361%	0.000%
(1, 0, 0)	0.000%	17.546%	1.326%	9.380%	11.592%	13.985%	0.000%	0.000%
(1, 0, 1)	0.000%	18.384%	0.922%	10.181%	11.114%	14.272%	0.000%	0.000%
(1, 1, 0)	0.000%	17.754%	2.225%	10.345%	10.480%	14.183%	0.000%	0.000%
(1, 1, 1)	0.000%	18.297%	1.925%	9.387%	12.153%	13.822%	0.000%	0.000%

RM# / (α, β, γ)	9(*)	10	11	12	13	14	15	16
(0, 0, 0)	0.000%	20.393%	41.270%	13.349%	10.774%	9.337%	27.047%	47.123%
(1, 0, 0)	0.000%	11.068%	16.863%	11.245%	9.581%	8.313%	16.994%	16.476%
(1, 0, 1)	0.000%	10.506%	16.081%	10.133%	9.110%	7.075%	17.581%	18.183%
(1, 1, 0)	0.000%	11.701%	17.216%	10.392%	9.437%	7.877%	15.744%	14.426%
(1, 1, 1)	0.000%	12.545%	18.650%	10.624%	8.889%	8.090%	16.408%	16.691%



(a)



(b)

Figure 5. Aggregated bandwidth utilization in firm real-time allocation. (a) is the sum of bandwidth utilization of extra large bandwidth RMs, i.e. RM1 and RM9; (b) is the sum of bandwidth utilization of small bandwidth RMs, i.e. RM2-8 and RM10-16.

1) *Soft Real-time Allocation*

Here we investigate the performance of various resource selection policies and dynamic replication strategies. The results of the over-allocate ratio in soft real-time allocation are shown in Table IV. The policy (1,0,0) performs better than the policy (0,0,0) in all cases. Although some collocation of resource selection policies can perform better on the over-allocate ratio, there is no noticeable improvement over the policy (1,0,0). On the other hand, the over-allocate ratios of dynamic replication strategies are all superior to that of the static replication strategy. Compared to the baseline strategy, strategy, Rep(1,8) can reduce the number of replicas transferred at a time and still yield a good performance. Moreover, strategy Rep(1,3) can save the storage capacity by decreasing the maximum number of replicas, with little degradation of performance. Therefore, strategy Rep(1,3) is of practical use as it takes into consideration the data traffic between the RMs and the storage capacity of the RMs, and still performs well. Fig. 6 shows the bandwidth utilization of RM1 and RM2 over time with the selection policy (1,0,0). It can be observed that dynamic replication indeed balances the workload of RMs as time goes by. The comparisons of the over-allocate ratio of each RM between static replication and strategy Rep(1,3) with selection policy (1,0,0) are shown in Fig. 7.

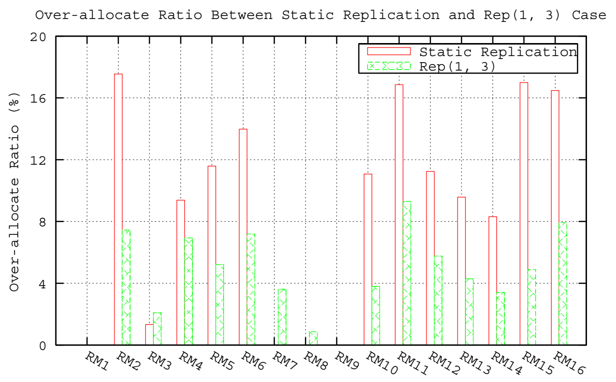


Figure 7. Comparison of over-allocate ratio of each RM between static replication and Rep(1, 3).

2) *Firm Real-time Allocation*

Based on the experience from the previous experiments, the experiments in the firm real-time allocation are conducted on the selection policies (0,0,0) and (1,0,0) only. The results of the fail rate in firm real-time allocation are shown in Table V. The policy (1,0,0) performs better than the policy (0,0,0) in all cases as in the soft real-time allocation. Similarly, the fail rates of dynamic replication strategies are all superior to that of the static replication strategy. The fail rate of strategy Rep(1,3) is still fairly acceptable as compared with that of baseline strategy and strategy Rep(1,8). In summary, the replication strategy Rep(1,3) with selection policy (1,0,0) can achieve 78% reduction on the over-allocate ratio and 86% reduction on the fail rate with respect to the static replication with selection policy (1,0,0).

3) *Destination Selection*

To further improve the bandwidth utilization, we also investigate the destination selection of replication strategy Rep(1, 3) in both soft real-time and firm real-time allocations. The three destination selection strategies investigated in this paper are given as follows: (1) Random selection strategy: the default strategy for all experiments; (2) Largest bandwidth first (LBF) strategy: only select the largest bandwidth RM, i.e. randomly select one of RM1 and RM9 in our configuration; and (3) Weighted selection strategy: RMs are selected randomly with probabilities according to their initial bandwidths. The experimental results are shown in Tables VI and VII. It can be observed that the weighted selection strategy is better in soft real-time allocation while the largest bandwidth first strategy is better in firm real-time allocation.

TABLE IV. AVERAGE OVER-ALLOCATE RATIO WITH DYNAMIC REPLICATION IN SOFT REAL-TIME ALLOCATION

Rep(N_{REP}, N_{MAXR}) \ (α, β, γ)	(0,0,0)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
Static replication	24.60%	9.77%	9.79%	9.54%	10.01%
Baseline	16.60%	1.44%	1.30%	2.86%	2.46%
Rep(1, 8)	15.67%	1.50%	1.47%	1.63%	2.40%
Rep(1, 3)	13.37%	2.17%	2.11%	1.38%	2.86%

TABLE V. AVERAGE FAIL RATE WITH DYNAMIC REPLICATION IN FIRM REAL-TIME ALLOCATION

Rep(N_{REP}, N_{MAXR}) \ (α, β, γ)	(0,0,0)	(1,0,0)
Static replication	15.62%	11.10%
Baseline	3.05%	1.20%
Rep(1, 8)	3.50%	1.17%
Rep(1, 3)	2.28%	1.50%

TABLE VI. AVERAGE OVER-ALLOCATE RATIO OF REP(1,3) WITH DIFFERENT DESTINATION SELECTION IN SOFT REAL-TIME ALLOCATION.

Destination selection \ (α, β, γ)	(0,0,0)	(1,0,0)
Random	13.37%	2.17%
LBW designated	10.41%	1.47%
Weighted	10.39%	1.28%

TABLE VII. AVERAGE FAIL RATE OF REP(1,3) WITH DIFFERENT DESTINATION SELECTION IN FIRM REAL-TIME ALLOCATION

Destination selection \ (α, β, γ)	(0,0,0)	(1,0,0)
Random	2.28%	1.50%
LBW designated	2.60%	1.20%
Weighted	3.05%	1.34%

VII. CONCLUSION

In this paper, we proposed a distributed file system with a storage QoS provision for cloud computing. Resource selection policies for resource management were investigated to improve bandwidth utilization and dynamic replication strategies were proposed to balance the workload of resource providers. We conducted experiments in real systems to study the performance on storage QoS with different resource selection policies and dynamic replication strategies. From the experimental results, we learned that resource selection policies can benefit bandwidth utilization, but they still have their limitations. Dynamic replication can further improve the bandwidth utilization by balancing the workloads of RMs. The strategy Rep(1,3) is of practical use as accounts for the data traffic between RMs and the storage capacity of RMs and still performs well. The replication strategy Rep(1,3) with selection policy (1,0,0) can achieve 78% reduction on the over-allocate ratio in soft real-time allocation and 86% reduction on the fail rate in firm real-time allocation with respect to the static replication with selection policy (1,0,0). In conclusion, the investigations on resource selection policies and dynamic replication strategies can facilitate the storage QoS provision in distributed file systems for clouds.

ACKNOWLEDGMENT

This work was supported in part by National Science Council, Taiwan, ROC, under Contract No. NSC-99-2221-E-001-004-MY3 and NSC-100-2218-E-001-006.

REFERENCES

[1] Block i/o controller [online]. Available: http://www.mjmwired.net/kernel/ Documentation/cgroups/blkio-controller.txt.

[2] Filesystem in userspace [online]. Available: http://fuse.sourceforge.net/.

[3] Kernel based virtual machine [online]. Available: http://www.linux-kvm.org/page/Main_Page.

[4] Lustre file system [online]. Available: http://wiki.lustre.org/index.php/ Main_Page.

[5] Qemu [online]. Available: http://wiki.qemu.org/Main_Page.

[6] Vmware [online]. Available: http://www.vmware.com/.

[7] P. Barham, et al., "Xen and the art of virtualization," in *Proc. of the 19th ACM Symp. on Operating systems principles*, SOSP '03, pages 164-177, New York, NY, USA, 2003. ACM.

[8] A. Batsakis, R. Burns, A. Kanevsky, J. Lentini, and T. Talpey, "Ca-nfs: A congestion-aware network file system," Trans. Storage, 5(4):15:1-15:24, Dec. 2009.

[9] D. O. Bigelow, et al., "End-to-end performance management for scalable distributed storage," in *Proc. of the 2nd Int.Workshop on Petascale data storage: held in conjunction with Supercomputing* '07, PDSW '07, pages 30-34, New York, NY, USA, 2007. ACM.

[10] Z. Dimitrijević and R. Rangaswami, "Quality of service support for real-time storage systems," in *Proc. of the Int. IPSI-2003 Conf.*, Oct. 2003.

[11] S. Elmohamed. Kosmosfs, http://code.google.com/p/ kosmosfs/.

[12] S. Ghemawat, H. Gobio, and S.-T. Leung, "The google file system," in *Proceedings of the 19th ACM Symp. on Operating systems principles*, SOSP'03, pages 29-43, New York, NY, USA, 2003. ACM.

[13] M. A. Gibney and N. R. Jennings, "Dynamic resource allocation by market-based routing in telecommunications networks," in *Proc. of the 2nd Int. Workshop on Intelligent agents for telecommunication applications*, pages 102-117, London, UK, 1998. Springer-Verlag.

[14] E. K. Lee and C. A. Thekkath, "Petal: distributed virtual disks," in *Proc. of the 7th Int. Conf. on Architectural support for programming languages and operating systems*, ASPLOS-VII, pages 84-92, New York, NY, USA, 1996. ACM.

[15] C. Lu, G. A. Alvarez, and J. Wilkes, "Aqueduct: Online data migration with performance guarantees," *in Proc. of the 1st USENIX Conf. on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002, USENIX Association.

[16] J. Montes, B. Nicolae, G. Antoniu, A. Sánchez, and M. S. Pérez, "Using global behavior modeling to improve qos in cloud data storage services," in *Proc. of the 2010 IEEE 2nd Int. Conf. on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 304-311, Washington, DC, USA, 2010.

[17] F. Mu, J. Shu, B. Li, and W. Zheng, "Multi-dimensional storage qos guarantees for an object-based storage system," in *Proc. of the 6th Int. Conf. on Computational Science*, Part III, ICCS '06, pages 687-694. Springer, May 2006.

[18] S. Narasimhamurthy, J. Morse, D. Golbourn, G. Umanesan, and M. Muggeridge, "Storage quality of service for high performance computer clouds," in *Proc. of UK e-Science All Hands Meeting 2010*, Sept. 2010.

[19] G. Neiger, A. L. Santoni, F. H. Leung, D. Rodgers, and R. Uhlig, "Intel virtualization technology: Hardware support for efficient processor virtualization," Intel Technology J., 10:167-178, 2006.

[20] D. Nikolow, R. Slota, and J. Kitowski, "Storage qos aspects in distributed virtualized environments," in *Proc. of the 1st Int. Conf. on Cloud Computing, GRIDs, and Virtualization*, pages 110{115. IARIA, Nov. 2010.

[21] K. Ranganathan and I. T. Foster, "Identifying dynamic replication strategies for a high-performance data grid," *in Proc. of the 2nd Int. Workshop on Grid Computing*, GRID '01, pages 75-86, London, UK, UK, 2001, Springer-Verlag.

[22] Y. Saito, S. Fr lund, A. Veitch, A. Merchant, and S. Spence, "Fab: building distributed enterprise disk arrays from commodity components," in *Proc. of the 11th Int. Conf. on Architectural support for programming languages and operating systems*, ASPLOS-XI, pages 48-58, New York, NY, USA, 2004. ACM.

[23] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *Proc. of IEEE 26th Symp. on Mass Storage Systems and Technologies*, pages 1-10, May 2010.

[24] R. G. Smith, “The contract net protocol: High-level communication and control in a distributed problem solver,” *IEEE Trans. Comput.*, 29:1104-1113, December 1980.

[25] B. Trushkowsky et al., “The scads director: scaling a distributed storage system under stringent performance requirements,” *in Proc. of the 9th USENIX conf. on File and storage technologies*, FAST'11, Berkeley, CA, USA, 2011. USENIX Association.

[26] V. Velusamy and A. Skjellum, “Quality of service support for grid storage environments,” in *Proc. of Int. Conf. on Grid Computing and Applications*, pages 134-140, June 2006.

[27] C.-M. Wang, H.-M. Chen, C.-C. Hsu, and J. Lee, “Resource selection strategies for a cnp-based resource management model,” in *Proc. of the 2008 IEEE Asia-Pacific Services Computing Conference*, pages 458-463, Washington, DC, USA, 2008. IEEE Computer Society.

[28] C.-M. Wang, C.-C. Huang, and H.-M. Liang, “Asdf: An autonomous and scalable distributed file system,” *in Proc. of the 2011 11th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 485-493, Washington, DC, USA, 2011. IEEE Computer Society.

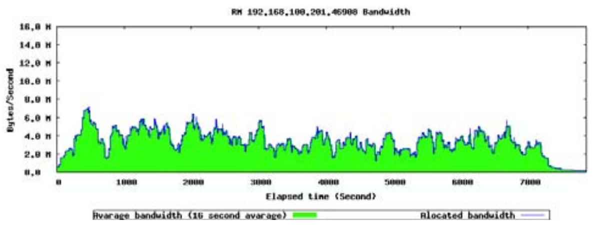
[29] Q. Wei, W. Lin, B. Veeravalli, and L. Zeng, “Qos-aware striping with replication model for object-based multimedia storage,” in *Proc. of the 4th Int. Workshop on Storage Network Architecture and Parallel I/Os*, pages 114-121. IEEE Computer Society, Sept. 2007.

[30] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, “Ceph: a scalable, high-performance distributed file system,” in *Proc. of the 7th Symp. on Operating systems design and implementation*, OSDI '06, pages 307-320, Berkeley, CA, USA, 2006. USENIX Association.

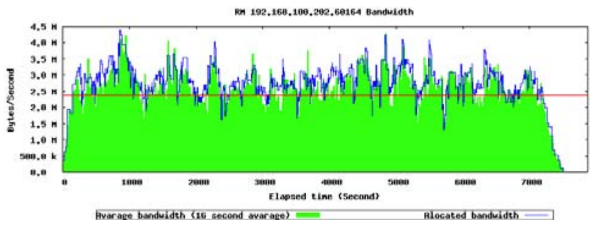
[31] W. W. Wilcke, et al., “Ibm intelligent bricks project: petabytes and beyond,” *IBM J. Res. Dev.*, 50:181-197, March 2006.

[32] O. Wolfson, S. Jajodia, and Y. Huang, “An adaptive data replication algorithm,” ACM Trans. Database Syst., 22(2):255-314, June 1997.

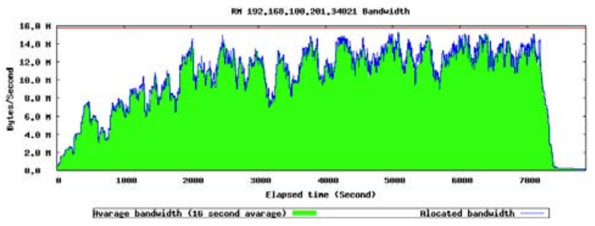
[33] J. C. Wu and S. A. Brandt, „Storage access support for soft real-time applications,” in *Proc. of the 10th IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 164-171, Washington, DC, USA, 2004. IEEE Computer Society.



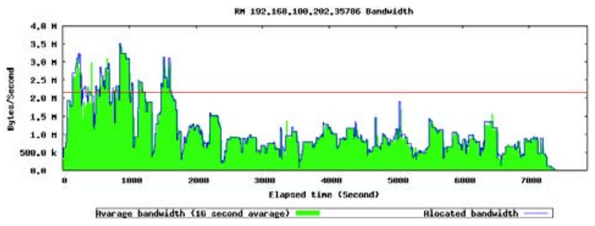
(a) RM1, static replication



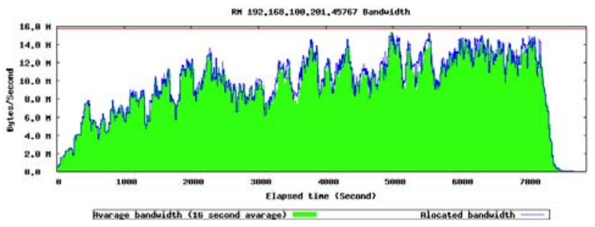
(b) RM2, static replication



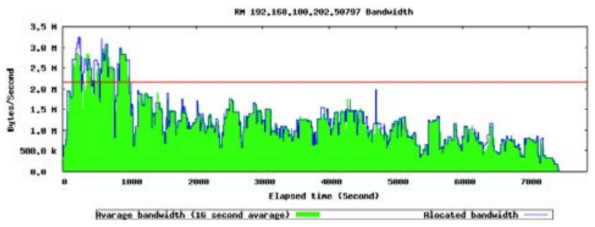
(c) RM1 , Rep(3, 8)



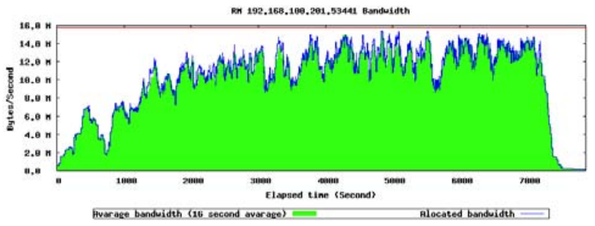
(d) RM2 , Rep(3, 8)



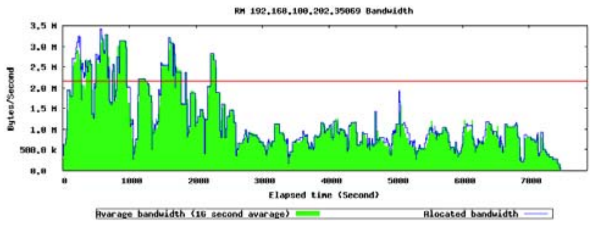
(e) RM1 , Rep(1, 8)



(f) RM2 , Rep(1, 8)



(g) RM1 , Rep(1, 3)



(h) RM2 , Rep(1, 3)

Figure 6. Bandwidth utilization of large bandwidth RM1 and small bandwidth RM2 with four dynamic replication strategies. The red solid line indicates the maximum available bandwidth assigned to the RM.