

# A Service-Oriented Approach to Storage Backup\*

Hao Cheng<sup>1</sup>, Yao H. Ho<sup>1</sup>, Kien A. Hua<sup>1</sup>, Danzhou Liu<sup>1</sup>, Fei Xie<sup>1</sup>, and Ynn-Pyng Tsaur<sup>2</sup>

<sup>1</sup>*School of EECS, University of Central Florida*  
{haocheng, yho, kienhua, dzliu, xiefei}@cs.ucf.edu

<sup>2</sup>*Symantec Corporation*  
anker\_tsaur@symantec.com

## Abstract

Various organizations face an explosive growth of data that must be protected and backed up. This challenge is made more difficult by the movement from stand-alone server backup to backup over the local area network (LAN) and by the need to automatically manage multiple backup servers efficiently for concurrent backup jobs. In this paper, we present a novel software approach to backup service, where application servers provide their unused resources to participate in a virtual backup environment. We implement a prototype with the following desirable features: single-instance storage to save storage and communication costs, data replication to enhance data availability, data redirection to avoid job resubmission, and intelligent scheduling to achieve workload balance. The experimental study shows that our approach is promising.

## 1. Introduction

Backups protect data from application and operating system corruption, storage corruption, network corruption, power outages, user errors, and natural disasters. With the explosive growth of data in recent years, storage backup becomes increasingly more challenging. In particular, the movement from a monolithic stand-alone server backup strategy to that of backups over the LAN, and the need to automatically manage multiple backup servers efficiently for concurrent backup jobs pose new challenges. To the best of our knowledge, most existing backup systems for organizations conform to the traditional client-server model. However, this model that uses dedicated backup servers has the following shortcomings:

- Low performance: A lightly-loaded backup server cannot offload work from a heavily-loaded backup server.
- Less robustness: The backup server becomes a single point of failure and can delay backup jobs.
- High costs: Using dedicated backup hardware incurs additional hardware, software, and operating costs.

To address the aforementioned issues, we make the following observations. Individual business units of a

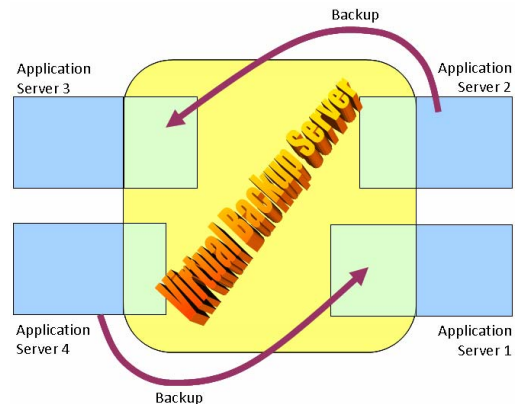


Figure 1: Sharing resource with applications

typical enterprise have their own application servers, and these servers are underutilized most of the time. In fact, the common practice is to have server capacity twice what is required to handle peak time. Our motivation, therefore, is to leverage otherwise wasted machine cycles for backup applications. This strategy is illustrated in Figure 1. It shows a novel software approach which relies on a community of servers to support a virtual backup system. In this scheme, the application servers donate part of their computing resources to the pool of backup nodes. When a backup job arises, some number of underutilized backup nodes are selected dynamically for the task (i.e., “stealing” otherwise wasted machine cycles from underutilized application servers). Since a currently lightly-loaded server is selected for a given backup job, high performance can be achieved. Furthermore, since any backup node can provide data backup service for any of the application servers, our design is fault tolerant. This approach also incurs less cost because the backup nodes share hardware with the application servers (i.e., no dedicated backup servers). The reduction in the number of computing servers simplifies IT operations, and therefore lowers the total cost of ownership.

The remainder of this paper is organized as follows. In Section 2, we examine some related work. The proposed system architecture is introduced in Section 3. In Section 4, we present the system prototype in details. The experimental results are discussed in Section 5. Finally, we conclude this paper in Section 6.

\* This research is partially supported by a grant from Symantec.

## 2. Related Work

In this section, we discuss some existing related work on storage backup. They can be roughly divided into four categories: client-server backup, peer-to-peer backup, grid computing, and massive storage.

Most existing commercial backup systems conform to the traditional client-server model. This model suffers from limited system utilization, high hardware cost, limited fault tolerance, and high administration cost. Our approach, as illustrated in Figure 1, is a new model to address the above limitations. More specifically, our approach aims to lower the cost of data protection while improving the performance and reliability of the backup service.

Peer-to-peer backup has received a lot of attention [6, 7, 8, 12, 16]. For example, Cox *et al.* have developed a peer-to-peer backup system, named *Pastiche* [7]. *Pastiche* chooses peers that share a significant amount of data to minimize storage overhead, and then exploits their excess disk capacity to perform peer-to-peer backup with no administrative costs. In addition, Cooper *et al.* [6] propose a mechanism for allowing peers to conduct peer-to-peer data trading to achieve reliability. However, such peer-to-peer backup systems require a large number of replicas to handle the dynamic nature of P2P system such that peers can join and leave at any time, and are not suitable for enterprise environments where servers are on in most of the time, and data security should be guaranteed.

Other efforts go to grid computing [11, 21, 23]. The widely-used Globus Toolkit 4 (GT4) [2] is an open-source software development toolkit for grid computing. Based on the data grid of GT4, Wan *et al.* [21] propose an approach to manage replicas of data on low cost storage media as simple as on a disk cache, and to discover, access, and manipulate data stored on tape media through the same interfaces that data grid provides for access to all types of storage systems. Nevertheless, grid computing is mainly for computing, and data grid is for sharing storage. That is, grid computing is not developed purposely for enterprises' continuous data protection.

Many efforts have also been made on massive storage [5, 9, 10, 13, 14, 15, 17, 18, 19, 20, 22]. OceanStore [17] is a global persistent data store designed to scale to billions of users. It provides a consistent and durable storage utility, and any server may create a local replica of any data. These local replicas facilitate faster access and robustness to network partitions, and also reduce network congestion by localizing access traffic. OceanStore also employs a Byzantine-fault tolerant commit protocol to keep consistency across replicas. However, those massive storage systems are not developed purposely for enterprises' data backup. On the other hand, our approach aims to automatically back up

with high scalability and flexibility, and to provide continuous data protection.

## 3. System Architecture

The proposed system architecture consists of the following six major components:

**Job Pre-processing Module.** This module supports three backup levels: full backup (complete dumps), differential backup (files changed since last full backup), and incremental backup (changed files since the last backup of any sort). The desired backup level and scheduled time for a given backup job are specified here. This module also cooperates with the catalog service module (discussed later) to achieve single instance storage [4] for storage saving.

**Backup Operation Module (BKOM).** The basic functionality of this module is to transfer data during backups and restores. In a backup operation, BKOM forwards the backup data from the job pre-processing module to a storage node. In our design, every machine in the system can be a storage node if it contributes its storage space.

**Job Scheduling Module (JSM).** Basically, the job scheduling module accepts four types of job requests: backup, restore, replication, and redirection job requests; makes scheduling decisions (i.e., when to schedule, and where to store) for them based on storage nodes' status; and forwards those decisions to the job monitor module.

**Job Monitor Module (JMM).** The job monitor is responsible for monitoring running jobs. A running job queue is maintained to track jobs' status. The job monitor cooperates with BKOM and the local storage manager (discussed shortly) to carry out various data transfer and storage operations. The job monitor also interacts with the job scheduling module that may need running job information to make scheduling decisions. After a backup job is done, the job monitor submits the corresponding replication job request to the job scheduler.

**Local Storage Manager Module.** This module enables machines to share their storage space for the backup service. The local storage manager cooperates with BKOM for data transfer during backup and restore. Besides responsible for replication, this module monitors the amount of free storage space in the local machine and triggers a redirection job request to avoid storage overflow.

**Catalog Service Module.** The catalog service is employed to store and retrieve summary information about the jobs, and files that are stored and replicated in various storage nodes. To enhance availability, we also maintain a replica of the original catalog on a second server.

## 4. System Prototyping

Following the above system architecture, we have developed a backup system with the following desirable features:

- Single-instance storage to save storage and communication costs.
- Data replication to enhance data availability.
- Data redirection to avoid job resubmission
- Intelligent scheduling to achieve workload balance.

In this section, we elaborate the detailed implementation of the job pre-processing module, job scheduling module, catalog service module, system operations, and system administration console.

#### 4.1. Job pre-processing

The job pre-processing module is based on Bacula [1], an open source network-based backup software. Bacula consists of the following five major components: Director Service, Console, File Service, Storage Service, and System Catalog. The desired backup level and scheduled time are specified in the definition of a backup job, which is controlled by the Directory Service. The backup or restore job is submitted via the Console. However, the Bacula Storage Service was developed to support the local disks or tapes instead of the distributed environment. We therefore modified the Storage Service to meet our needs.

It is observed that different users may back up some files that have identical content. Maintaining duplicate copies for each user in the backup system is a waste of disk space. To achieve single instance storage [4] for storage saving, we also modify the File Service accordingly, such as comparing signatures and generating backup job header (including files basic attributes). To detect whether two files are identical or not, we use the SHA-512 signatures instead of bit-by-bit comparison to reduce the computation and communication costs. To the best of our knowledge, there is no signature generation method that can completely guarantee that if two arbitrary files are different, their signatures are different as well. Although our approach is not theoretically guaranteed, it is still acceptable for backup applications. After generating the signatures, we send these signatures to the catalog service for signature comparison, and then we can decide which files are not necessary to be re-transmitted because they are already backed up in the system. After signature comparison, the catalog is updated accordingly.

#### 4.2. Job scheduling

Different types of job requests are submitted to JSM. JSM queues the incoming job requests and determines for each one, when to start this job and which storage node is

chosen to start the job. JSM tries to balance the workload and storage utilization among all the storage nodes.

There are four different types of job requests:

1. Backup, issued by the backup application. The scheduler determines which destination storage node is assigned to keep all the files of this job.
2. Restore, issued by the backup application. As our system may have multiple copies of the files requested by a job, the scheduler decides which node is chosen to send back all these files.
3. Replication, issued by the job monitor. The scheduler decides a replica is created in which destination node from which source node.
4. Redirection, issued by the job monitor. The job of the scheduler is to figure out the new destination node to store the remaining files of the redirected backup job.

The JSM accepts requests from different sources and makes decisions for them based on the current workload. The job monitor informs the selected nodes to set up the corresponding data transfer connections. We discuss job scheduling in details in the following subsections.

##### 4.2.1. Local resource monitor

A local resource monitor runs on a storage node. The monitor regularly collects the status information of the machine, including disk space usage and aggregate data transfer rate on its network ports. It periodically sends the information to the Resource Information Listener in JSM. Thus, JSM can maintain an up-to-date image of all the storage nodes, based on which scheduling decisions are made.

##### 4.2.2. Job scheduling module

Figure 2 shows the architecture of JSM and its major components are listed as follows.

**Internal Queue (IQ).** The IQ holds the requests which could not be scheduled right away as there are no nodes currently available to take the job.

**High Priority Queue (HPQ).** The queue holds requests of backup, restore, and also some replication requests that are promoted from the Low Priority Queue.

**Low Priority Queue (LPQ).** The LPQ holds replication requests. Only when HPQ is empty, the job entries in the front of LPQ can be promoted to HPQ in order.

**Job Request Receiver (JRR).** A job request is packed in a socket packet and submitted through the network to the JRR. There are multiple working threads waiting in the thread pool. Once a request arrives, one of the waiting threads wakes up and parses the received packet, which is transformed into the internal representation format of a job. Based on the job type, the request is inserted into either HPQ or LPQ as follows:

1. Backup: this type of requests is appended to the end of HPQ.
2. Restore: the restore jobs are considered more urgent compared to the backup jobs. This type of request is

inserted at the first position in the HPQ after all the restore and redirection jobs already in the queue.

3. Replication: this type of requests is appended to the end of the LPQ.
4. Redirection: this type of request is most urgent and is inserted at the first position in HPQ behind all redirection jobs already in the queue.

In two cases, JRR triggers and wakes up the scheduler as the new request arrives:

1. If currently there are no job requests in the queues, the scheduler is waiting for new jobs.
2. If the new arrived request is a redirection or restore job, and the scheduler is waiting as all the nodes are unavailable, JRR informs the scheduler to pick up and make a decision for the new request right away.

**System Status Information (SSI).** This component maintains status information about each node, including the disk quota, current disk utilization, data rate limit, current aggregate transfer rate and workload information (e.g., the number of running and waiting jobs at the node).

**Resource Information Listener (RIL).** The local resource monitor in each node regularly reports its own status information to RIL. If the RIL does not hear any update from a node, it actively contacts that node to determine if the node is still available. Moreover, the RIL constantly checks with the job monitor to catch the workload information of all the nodes. The RIL triggers the scheduler if the new status information reveals that a storage node becomes available. In response, the scheduler looks into the queues and dispatches the job if possible.

**Scheduler.** The scheduler dequeues the entries in IQ and HPQ and makes decisions for them properly. Based on the status information, the scheduler balances the workload by avoiding assigning too many jobs to a node.

Each individual storage node has a pre-defined data rate limit, which is set when the node joins the system. If the expected data transfer rate for a storage node exceeds its limit, this node is considered unavailable for backup and replication jobs.

The scheduler checks whether there are redirection or restore job requests in the system. If there are, the scheduler dequeues a job entry and picks up the storage node that has the best serving capability to perform this job. If there are no redirection and restore requests in the queue, the scheduler checks whether there are any job requests to back up files. If that is the case, the scheduler exams each storage node to see whether it is available. If there are nodes available, the scheduler picks one to serve the request. If HPQ becomes empty, replication requests in LPQ are promoted to HPQ. The scheduler continues to make decisions for the requests until there are no jobs in the queues or all the nodes are currently unavailable. Under these situations, the scheduler pauses and enters the waiting mode until either JRR informs that there are

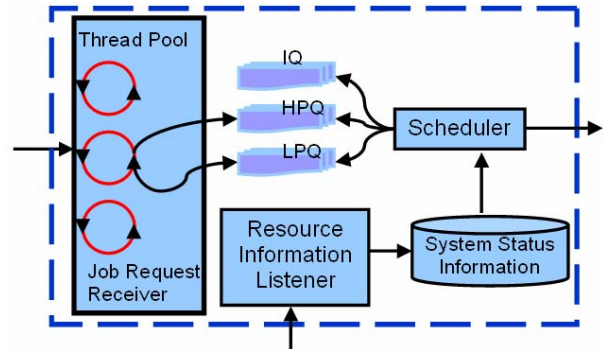


Figure 2: Job scheduling module

new requests, or RIL informs that some nodes become available.

### 4.3. Catalog Service

In this section, we discuss which tables are created and maintained in the catalog service, and how to enhance availability.

The catalog service maintains three major database tables: *Job*, *JobReplication*, and *JobFile*. The *Job* table contains one record for each job, recording each job's ID, type, status, and other related information. The job type can be backup, restore, replication or redirection; the job status can be waiting, running, blocked, succeeded or failed. The *JobReplication* table contains one entry for each replication job, and identifies its corresponding backup or redirection job ID. The *JobFile* table contains one entry for each file in a given job, and maintains the signature of each file already backed up to facilitate single instance storage. In addition, local file catalogs are also created and kept in storage nodes during system operations, and are used to reduce the workload of the catalog service.

In our prototype system, we use MySQL's replication mechanism to enhance availability. MySQL replication is based on the master server keeping track of all changes to the databases (e.g., inserts, updates, and deletes) in its binary logs [3]. Each slave server receives from the master the saved updates the master has recorded in its binary log, so that the slave can execute the same updates on its copy of the data.

### 4.4. System Operations

There are four distinct operations: backup operation, restore operation, redirection operation, and replication operation in our prototype. The user can issue a backup request to backup a set of files, or restore request to recover a set of files. When replicas are requested, the replication operation duplicates the files on multiple storage nodes more better data availability. Redirection is system operations. It occurs automatically when a backup

job encounters a storage node overflow. In this case, the files being backed up are redirected to another storage node, making the overflow transparent to the user. We discuss the details of these four operations in this section.

#### 4.4.1. Backup Operation

The backup operation is illustrated in Figure 3. BKOM acts as a relay which forwards the data to the remote storage node. To initialize the backup operation, the corresponding BKOM submits the backup job request to the JSM. The JSM sends the scheduling decision to the JMM. The scheduling decision contains the IP addresses and ports of BKOM and the local storage manager, respectively. After JMM establishes the connection between BKOM and the local storage manager, the data is transferred from BKOM to the destination local storage manager and finally stored in the corresponding storage node. The data transfer occurs in a push manner. That is, BKOM pushes the data to the local storage manager.

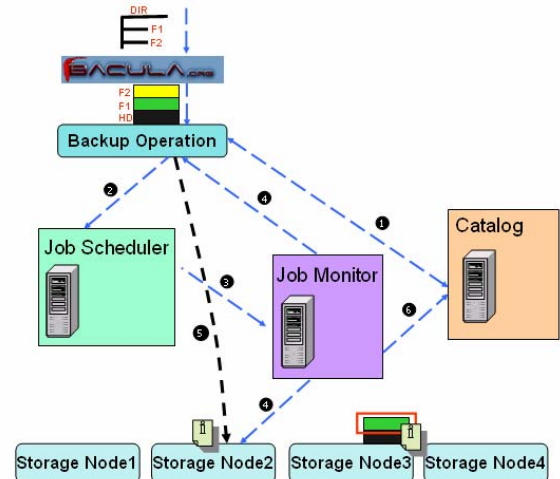


Figure 3: Backup with single instance storage

#### 4.4.2. Restore operation

The restore operation (see Figure 4) is the reverse of the backup operation. After receiving the request to restore the data of a previous backup job, BKMO forwards the request to the scheduler. JMM picks the best storage node which contains the local file catalog of the requested backup job. The selected storage node sends the local file catalog to the BKOM. Since the local file catalog contains the physical address (i.e., the storage node address and file path) of all the files of the backup job, BKOM can pull the file data from the corresponding storage nodes with the local file catalog of the requested backup job. If a file has more than one physical instance, BKOM randomly picks one. If this randomly picked one does not work, the BKOM switches to other instances.

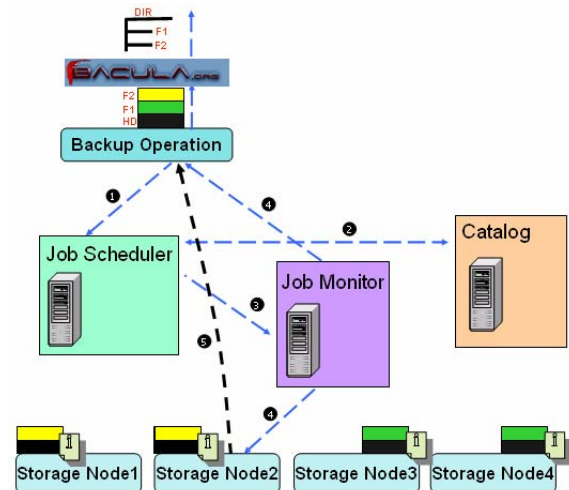


Figure 4: Restore with single instance storage

#### 4.4.3. Replication Operation

After a backup job succeeds, JMM submits a replication request to JSM to make a replica (see Figure 5). The replication job has secondary priority in the JSM and will be scheduled after the backup and restore jobs. JSM initiates the replication operation by selecting the storage node which contains the files backed up earlier as the source node, and choosing another storage node to serve as the destination node of the replication. The JMM then notifies the source and destination storage nodes to start the replication process.

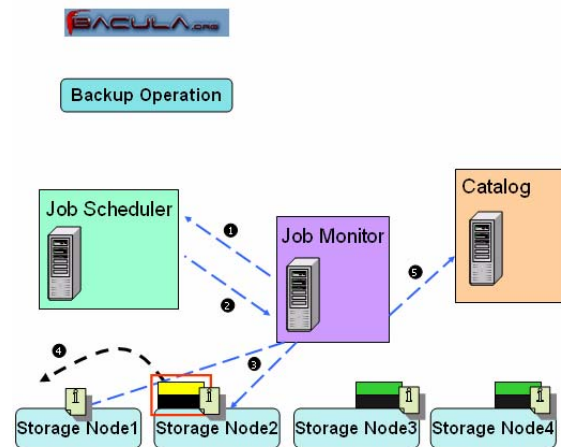


Figure 5: Replication

#### 4.4.4. Redirection Operation

For backup applications, the size of the entire backup data may be unknown ahead of time. Therefore, it is possible to select a storage node that does not have enough free storage space to hold the entire backup data. This causes the backup job to fail due to the storage overflow. To address this problem, we propose a data redirection scheme. When a storage node does not have

enough space to store the entire backup data, the un-stored data is transferred to another storage node with a plurality of available storage. This approach makes the storage overflow transparent to the users and saving the costs of data retransmission.

The redirection operation is implemented as follows. The local storage manager monitors the free space of the local machine during the backup operation. When the size of free space reaches a specified threshold, the local storage manager sends a message to pause BKOM from pushing data to it. Meanwhile, the local storage manager sends a redirection request to JSM. The redirection request is handled with highest priority and will be processed right away. Usually, JSM chooses the storage node with largest free storage space as the alternative. JMM establishes connection between BKOM and the new assigned storage node. BKOM then resumes the data transfer. After the backup job is done, the catalog service records information to reflect that the backup job currently has two segments in two different storage nodes. To combine the two segments to one storage node, the new (alternate) storage node pulls the segment from the overflow storage node and combines the two segments together. This combination operation is optional and would be considered only if the alternate storage node has enough space. It is possible that multiple redirections happen during a backup operation. In such case, the data is redirected to multiple storage nodes.

#### 4.5. System Administration Console

The system administration console consists of three components – *Administrator Login*, *System Configuration*, and *System Monitor*. System administrators need to login at the *Administrator Login* window. Once logging in, administrators can configure and define thresholds for backup, restore, or replication in the *System Configuration* window. The system performance and status will be available at the *System Monitor* window. The system status is updated by the job monitor. First, the job monitor updates a simplified log file named Job Dump. Job Dump (e.g., jobdump1.txt) contains information for current running jobs as follows — job ID, job type (i.e., backup, restore, replication, or redirection), job size, job transferred size, source, destination, and new destination (for redirection only).

When a new job is submitted to the system, the job monitor calls RRDbuild to create a *Round Robin Database* (RRDB) for the new job ID (e.g., job\_1.rrd) using RRDtool (version 1.2.0). The job monitor periodically observes the status of on-going jobs within the system. When the status (i.e., transferred size or

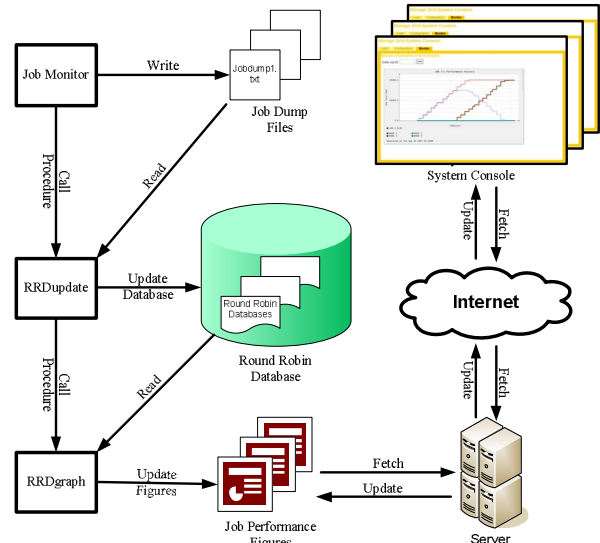


Figure 6: Updating system information

destination node) of an on-going job is changed, the job monitor calls an update procedure (i.e., RRDupdate) to update the job’s RRDB. Once the update procedure finished, the RRDgraph is called to create or update the performance graph for the updating job (see Figure 6).

In the *System Monitor* window, the administrator can enter the job ID to show the current status of this on-going job. The job ID is sent to the Web server. The Web server fetches the corresponding job figure and sends the job figure back to the system administrator console. This console continues to fetch and update the figure until either the job is completed or administrator enters another Job ID (see Figure 6).

#### 5. Performance Study

To evaluate the proposed framework for storage backup, we perform experiments on the prototype system presented in Section 4. The experimental setting consists of six machines. Two machines acted as servers; and the other four were storage nodes. The detailed hardware and software configurations are illustrated in Figure 7. Our workload is as follows. We sequentially submitted seven jobs: three backup jobs, two replication jobs, and two restore jobs. Three data sets were used in our experiments:

- Data Set 1 with four files: |F1|=100 Mbytes, |F2|<1 Mbytes, |F3|<1 Mbytes, and |F4|=1.4 Mbytes
- Data Set 2 with two files: |F1|=100 Mbytes and |F4|=1.4 Mbytes

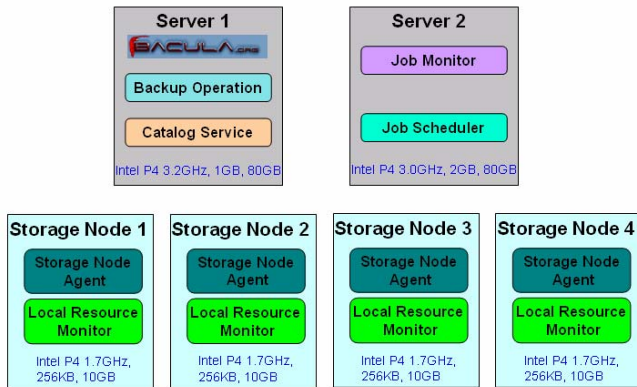


Figure 7: Experimental setting

- Data Set 3 with two files: |F4|=1.4 Mbytes and |F5|=200 Mbytes

Figures 8-14 illustrate the data transmission as time elapsed for the aforementioned seven jobs, respectively. For comparison purpose, the red curves in all these figures show the amount of data transmitted without the single instance storage feature. Specifically, Figure 8 illustrates the results for the first backup job—Job 1 to back up all files in Data Set 1. The job size is about 100 Mbytes. After we submitted Job 1 through the job pre-processing module in Server 1, the job scheduler in Server 2 chose Storage Node 4 as the destination in this case. Since initially no file was backed up in the whole system before Job 1, all files (i.e., F1, F2, F3 and F4) were stored with the job header (including each file’s basic attributes such as file name, type, ownership, and access permissions) in Storage Node 4. The sky blue curve shows the data transmission from Server 1 to Storage Node 4 as time elapsed, while the red curve shows the data transmitted without the single instance storage feature. These two curves overlap for most part under Job 1, and the zigzag pattern is due to sampling.

After Job 1 was completed, we submitted the second backup job—Job 2 to back up all files in Data Set 2. Then, the signatures of F1 and F4 were used to query the catalog for signature comparison. Since Job 1 already backed up F1 and F4 in the system, it was not necessary to back up them again. As illustrated in Figure 9, only the header of Job 2 (less than 1 Mbytes) was transmitted and stored on Storage Node 4. That is, network communication and disk storage overheads are significantly reduced for Job 2. It is also worthy to point out that Storage Node 4 was again selected to be the destination due to the intelligent job scheduler: try to keep the job header to the same storage node of the data if applicable.

Afterwards, we submitted the third backup job—Job 3 to back up all files in Data Set 3. In order to trigger the redirection operation as discussed in Section 4.4.4, we manually set the threshold to 150 Mbytes. Data Set 3

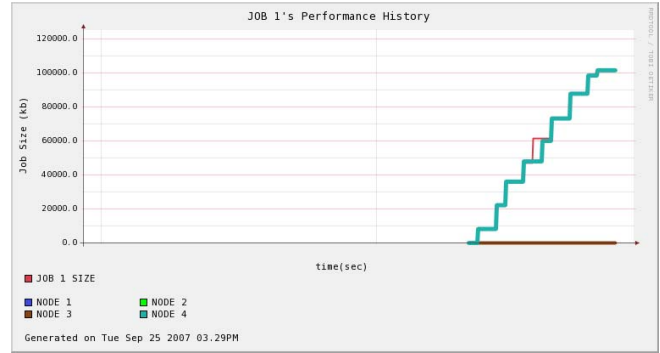


Figure 8: Job 1 performance

consists of two files: F4 (already backed up in Job 1), and F5 (200 Mbytes which is more than the specified threshold of 150 Mbytes, need to be backed up). Figure 10 shows the results of this job. Initially, the assigned storage node was Storage Node 1 (see the blue curve in Figure 10). After 150 Mbytes (i.e., the specified threshold) data were transmitted to Storage Node 1, a redirection operation was triggered. Then, Storage Node 3 was assigned by the job scheduler to take over Job 3: established a new connection, switched the stream, and finally combined the two data parts (see the violet curve in Figure 10). If without the data redirection mechanism, this job would have to be re-submitted, which incurs extra network communication cost.

To enhance data availability, we also submitted two replication jobs. That is, Job 4 was to replicate the data for Job 1 (see Figure 11), and Job 5 was for Job 2 (see Figure 12). Specifically, the data set of Job 1 (i.e., Data Set 1) was stored in Storage Node 4, and Storage Node 2 (see green curve in Figure 11) was picked by the scheduler for replication with about 100 Mbytes data transmitted and stored. On the other hand, Figure 12 shows that for Job 5 less than 1 Mbytes of data were replicated from Storage Node 4 to Storage Node 2.

Finally, we performed two restore jobs; i.e., Jobs 6 and 7 were to restore the data sets of Job 2 and Job 3, respectively. We note that the data set of Job 2 is Data Set 2 with two files F1 and F4. Due to the replication, Data Set 2 had two copies (in both Storage Nodes 4 and 2). The scheduler chose Storage Node 2 for the restore, as shown in Figure 13. Furthermore, Data Set 3 consists of F4 (stored in Storage Nodes 4 and 2) and F5 (stored in Storage Node 3). Therefore, to restore this data set, F4 was retrieved from Storage Node 4, and F5 was from Storage Node 3 (see Figure 14).

Our experimental results indicate that storage and communication costs can be significantly reduced with single instance storage, data availability can be strengthened with data replication, job resubmission can be avoided with data redirection, and workload balance can be achieved with the intelligent scheduling technique.

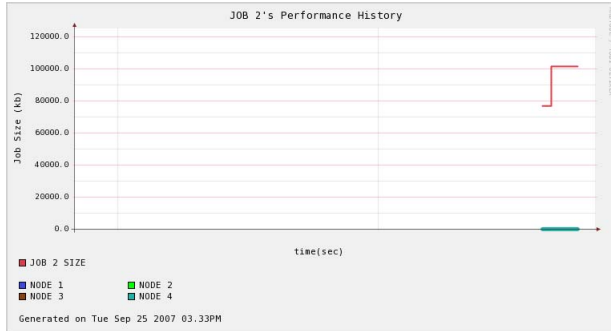


Figure 9: Job 2 performance

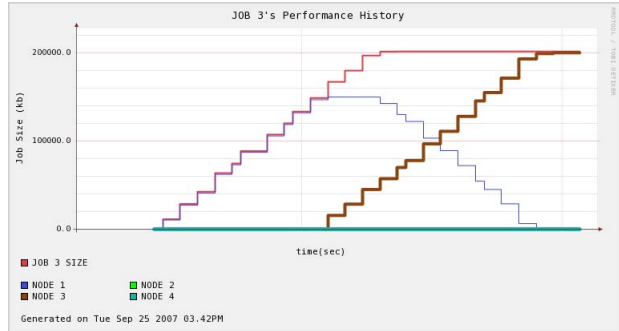


Figure 10: Job 3 performance

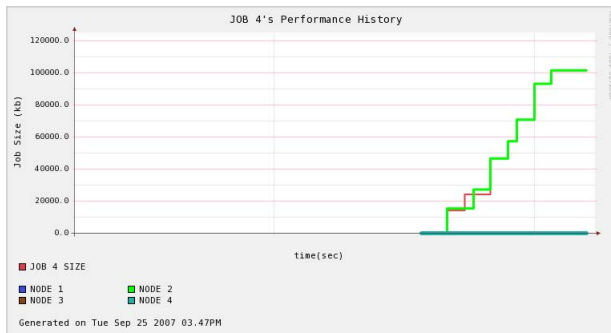


Figure 11: Job 4 performance

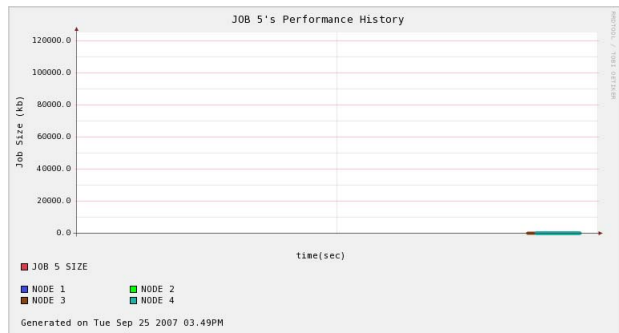


Figure 12: Job 5 performance

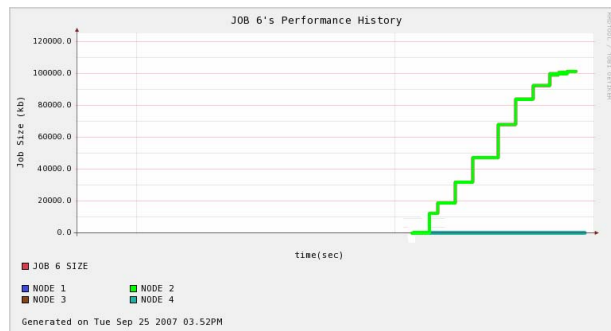


Figure 13: Job 6 performance

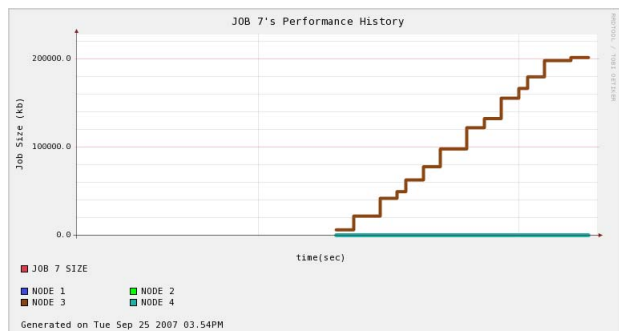


Figure 14: Job 7 performance

## 6. Concluding Remark

The contribution of this paper is a new service-oriented framework for storage backup. This software approach offers organizations a scalable, robust, efficient, and economical solution to doing backups and restores across a distributed organization—whether one that has multiple servers in one campus or distributed among remote offices. The experimental results confirm that our prototype can significantly reduce the storage and communication costs, and achieve better fault tolerance and workload balance. Future research may extend our approach to support an Internet-based backup service for individuals.

## References

- [1] Bacula. <http://www.bacula.org>.
- [2] Globus Project. <http://www.globus.org>.
- [3] MySQL. <http://www.mysql.com>.
- [4] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur. Single instance storage in Windows 2000. In *Proceedings of the 4th USENIX Windows Systems Symposium*, pages 13-24, 2000.
- [5] S. Chandra, A. Gehani, and X. Yu. Automated Storage Reclamation Using Temporal Importance Annotations. In *Proceedings of the International Conference on Distributed Computing Systems*, 2007.
- [6] B. F. Cooper and H. Garcia-Molina. Bidding for Storage Space in a Peer-to-Peer Data Preservation System. In

- Proceedings of the International Conference on Distributed Computing Systems*, pages 372-381, 2002.
- [7] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: making backup cheap and easy. In *Proceedings of the 5th symposium on Operating systems design and implementation*, pages 285-298, 2002.
- [8] L. P. Cox and B. D. Noble. Samsara: honor among thieves in peer-to-peer storage. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 120-132, 2003.
- [9] S. Cranage. OpenSMS. In *Proceedings of the IEEE/NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 169-178, 2005.
- [10] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 29-43, 2003.
- [11] T. Kosar and M. Livny. Stork: Making Data Placement a First Class Citizen in the Grid. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 342-349, 2004.
- [12] D. Li, X. Lu, B. Wang, J. Su, J. Cao, K. C. C. Chan, and H. V. Leong. Delay-Bounded Range Queries in DHT-based Peer-to-Peer Systems. In *Proceedings of the International Conference on Distributed Computing Systems*, 2006.
- [13] Q. Lian, W. Chen, and Z. Zhang. On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 187-196, 2005.
- [14] S. Liang, S. Jiang, and X. Zhang. STEP: Sequentiality and Thrashing Detection Based Prefetching to Improve Performance of Networked Storage Servers. In *Proceedings of the International Conference on Distributed Computing Systems*, 2007.
- [15] N. Mandagere, J. Diehl, and D. Du. GreenStor: Application-Aided Energy-Efficient Storage. In *Proceedings of the IEEE Conference on Mass Storage Systems and Technologies*, pages 16-29, 2007.
- [16] I. Osipkov, P. Wang, N. Hopper, and Y. Kim. Robust Accounting in Decentralized P2P Storage Systems. In *Proceedings of the International Conference on Distributed Computing Systems*, 2006.
- [17] S. C. Rhea, P. R. Eaton, D. Geels, H. Weatherspoon, B. Y. Zhao, and J. Kubiatowicz. Pond: The OceanStore Prototype. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 1-14, 2003.
- [18] S. Seshadri, L. Liu, B. F. Cooper, L. Chiu, K. Gupta, and P. Muench. A Fault-Tolerant Middleware Architecture for High-Availability Storage Services. In *Proceedings of the IEEE International Conference on Services Computing*, pages 286-293, 2007.
- [19] S. Uttamchandani, K. Voruganti, R. Routray, L. Yin, A. Singh, and B. Yolken. BRAHMA: Planning Tool for Providing Storage Management as a Service. In *Proceedings of the IEEE International Conference on Services Computing*, pages 1-10, 2007.
- [20] A. Verma, D. Pease, U. Sharma, M. Kaplan, J. Rubas, R. Jain, M. V. Devarakonda, and M. Beigi. An Architecture for Lifecycle Management in Very Large File Systems. In *Proceedings of the IEEE/NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 160-168, 2005.
- [21] M. Wan, A. Rajasekar, R. Moore, and P. Andrews. A Simple Mass Storage System for the SRB Data Grid. In *Proceedings of the IEEE/NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 20-25, 2003.
- [22] R. W. Watson. High Performance Storage System Scalability: Architecture, Implementation and Experience. In *Proceedings of the IEEE/NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 145-159, 2005.
- [23] H. M. Wong, Y. W. Wang, H. N. Yeo, D. Wang, Z. Li, K. H. Leong, and K. L. Yong. Dynamic Storage Resource Management Framework for the Grid. In *Proceedings of the IEEE/NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 286-293, 2005.