Accelerate Data Sharing in a Wide-Area Networked File Storage System[⋆]

Kun Zhang, Hongliang Yu, Jing Zhao, and Weimin Zheng

Department of Computer Science and Technology
Tsinghua University, Beijing 100084, China

Abstract. Up to now, more and more people use Internet storage services as a new way of sharing. File sharing by a distributed storage system is quite different from a specific sharing application like BitTorrent. And as large file sharing becomes popular, the data transmission rate takes the place of the response delay to be the major factor influencing user experience. This paper introduces strategies used to accelerate file sharing with low bandwidth consumptions in a deployed wide-area networked storage system - *Granary*. We use a popularity and locality sensitive replication strategy to put files closer to users that request it frequently. The *Hybrid* server selection scheme and the *Remote Boosting* replication mechanism are also presented. Experimental results show these methods offer better sharing speed and cost less network bandwidth than conventional caching schemes.

1 Introduction

The improvement of computer hardware and network infrastructure is offering more bandwidth and storage spaces, making it possible to distribute files as large as gigabytes via the Internet. While people storing more data through network, the need for sharing data like music, home video or picture albums is growing too. Email was used as a sharing tool before, which was inconvenient for its notable delay and limitation on file sizes.

Up to now, more and more people use Internet storage services as a new way of sharing. For example, commercial on-line storage services like Box.net and Omnidrive is offering more than 1GB storage space for each user. Sharing based on a storage system is quite different from specific file-sharing applications like BitTorrent and eMule. By uploading the file to storage servers, users does not need to keep on-line to share the data. The quality of service is also guaranteed by the infrastructure of the storage system, not end users. This lead to different choices of data replication and caching schemes.

As audio and video sharing becomes usual and popular, the data transmission rate takes the place of the response delay to be the major factor influencing user experience. From another aspect, as the file size grows, bandwidth consumption

[⋆] This work is supported by the National Natural Science Foundation of China under Grant No. 60603071 and 60433040 is becoming more critical than before. OceanStore [1] propose a introspection mechanism that can be used for cluster recognition or replica management, which could be a potential solution to this issue, but unfortunately there is not any further study or implementation for this mechanism. As OceanStore's prototype, Pond [2] uses a simple caching mechanism to accelerate archive data retrieval. Similarly, when a PAST node routes a file for a insertion or retrieval operation, it caches the file on its local storage [3]. However, caching is a passive scheme and is confined by the routing path of the original file. In Pangaea [4], nodes that have low network latency to each other are clustered into groups, and an aggressive replication mechanism is used to put replicas in the client's local nodes. However, it is not explained how nodes are clustered or how a client find a local node, which is very important. Moreover, the systems above do not tackle the performance problem of sharing large files sizing megabytes or even gigabytes, especially the transfer rate of downloading them. Therefore, it is still challenging to accelerate sharing speed with low bandwidth consumption in such systems.

In this paper, we introduce a distributed storage system which provides reliable storage service to over 500 users at present. Users backup their files and share some of them with others. We implement a replication mechanism to solve the problems brought by file sharing. It evaluates the popularity of files, clusters clients that are close to each other, makes more copies of popular files on the nodes that are closest to the most demanding users, thus decreases unnecessary network traffic, reduces network congestion on the critical links and finally improves download speed. It has two characteristics:

– *It's popularity sensitive.* It records the requests for every file, analyzes popularity, and propagates more replicas for more popular files. Popularity sensitivity can help relieve the impact of requests for popular files.
– *It's locality sensitive.* It detects and analyzes the location of storage nodes and clients, places file replicas closer to the users that request it frequently, thus decrease unnecessary inter-domain traffics and reduce network congestion on the critical links.

2 Background

Our distributed storage system is named as *Granary*.It is composed of dedicated nodes at a global scale and provides paid storage services to end users. It utilizes *PeerWindow* [5] as the node collection algorithm and *Tourist* [6] to route messages to proper nodes, on top of which a highly available DHT [7, 3] is built. The DHT stores DHT objects. A DHT object consists of a key and a value, and comes with a 128-bit hash value, which is the MD5 hash of its key. Every node is assigned a unique 128-bit key called nodeId, which can be the hash value of a per-machine signature such as MAC address. A DHT object is replicated and stored on the closest nodes in terms of nodeId, which are probably disperse in terms of physical location. This provides high availability to DHT objects.

Unlike PAST, *Granary* doesn't directly store files in DHT. Instead, it stores and replicates files in the upper layer and puts only the locations of replicas (which is called the *replica list*) and the meta data of files in the DHT. The high availability of DHT guarantees that the meta data can be accessible despite of network disconnections or node failures. A *Granary* server acts not only as a DHT node, which routes messages for other nodes and stores DHT objects that are mapped to it, but also as an upper-level storage server which clients access directly for file uploading and downloading. This choice of design allows us to develop a more flexible replication algorithm that are not constrained by the binding of nodeIds and objectIds in the DHT. Since trivial operations like message transmission and meta data read/write are all handled by the DHT layer, we are able to focus on the replication strategy itself.

Most contributive systems [7, 3, 1, 2] divide files into smaller blocks and spread blocks among nodes. The client connects to the closest node, which routes queries and data for it. This is a reasonable choice because they are made of contributive and weak nodes which tend to go off-line at any time. Dividing files into blocks and replicating these blocks ensures the availability of file even if some nodes become unavailable. However, *Granary* stores full copies of files on dedicated nodes. These nodes are more powerful and stable, but much less in quantity, compared with contributive systems. So *Granary* doesn't need to divide files to ensure availability. Additionally, storing full copies eliminates the need to communicate with many nodes simultaneously for file retrieval, thus reduces the traffic for block searching overhead.

Files in *Granary* are immutable, but a user can upload a new version of his file with the same file name. Different versions of a file are distinguished by version number. The servers do not delete the older versions immediately after a new version has been uploaded, because some other clients may be downloading the old one at the moment. The coexistence of older and newer versions will only long for a while, because only the newest version is visible to newly arriving clients, so that the older versions can be deleted after remaining download sessions have finished. When the user deletes a file, its meta data is removed from DHT, and the accommodating servers will delete their local replicas after remaining download sessions have finished.

In following sections, we will first describe some fundamental designs which will be later used. Then, strategies to accelerate sharing in *Granary* are presented. Furthermore, experimental results of these strategies will be shown at last.

3 Fundamental design

3.1 Caching vs replication

Both caching and replication are widely used to improve the performance of distributed systems [8, 7, 3]. Although they are very similar to each other, they do have differences. Firstly, caching passively retains data that may be used for the next time, while replication may actively create copies of data; Secondly, the data cached is volatile while replica is persistent; Finally, caching usually introduces no extra cost besides space occupation, while replication may bring extra cost, e.g. network traffic. We choose replication to accelerate the file retrieval in *Granary* for the following reasons:

- Replication is active, so we can do predictions and make copies proactively;
- Even if we use a caching method, replication is still needed to provide availability, and it would be a waste of space to keep both replicas and cache;
- Caching is used in systems where the data is transmitted via more than one nodes before reaching the client. Although caching does not bring extra network traffic, data routing and forwarding do. In *Granary*, clients download data directly from servers. If properly designed, it will not cause more network traffic with replication than a routing-and-forwarding system with caching.

3.2 Network distance

Sharing in *Granary* is boosted by the optimal placement of data replicas. To achieve it, we must know network distances in the system first. Usually, network distance between two hosts can be measured by round-trip time (RTT), trace-route hops, DNS name and IP address.

Trace-route reveals the underlying topology of the routers. However, it will brings too much extra traffic if we trace-route every requesting client. M. Andrews [9] grouped hosts into different autonomous systems (*ASes*) that have similar round-trip time (RTT) between them. He organized all known hosts by an *IP tree*, in which each node represents an IP prefix. With each server logs every file transmission to and from any other hosts, the similarities of RTT between leaf nodes are calculated. If the difference is below a threshold, these leaf nodes are merged and folded up to their parent. The result is a tree with its leaves representing different autonomous systems.

This method only considered IP address which may encountered a problem when an autonomous system includes several IP prefixes. For example, our campus network includes two network prefixes: (59.66.0.0/16) and (166.111.0.0/16), but they will never be clustered in such an IP tree. However, they are both assigned to the same DNS name suffix (*.ip.tsinghua.edu.cn*). To solve it, we introduce another data structure called the *domain tree*, with a similar structure as the DNS system. The *domain tree* records the DNS names of every clients. Except for using DNS name suffixes instead of IP prefixes, the *domain tree* is the same as the *IP tree*. Therefore the same clustering process can be applied to it.

In *Granary*, The IP distance $d_{ip}^{1,2}$ between hosts H_1 and H_2 is defined as the height of the smallest subtree that contains both the cluster leaf of H_1 and that of H_2. The height is zero based, and if the cluster leaf of a host doesn't exist, the default distance will be set to 24, as if a leaf is created for the host. If H_1 and H_2 both have DNS names, the domain distance $d_{domain}^{1,2}$ is defined in the same way in the domain tree. Otherwise, $d_{domain}^{1,2}$ will be set to the height of the domain tree. We divide d_{ip} and d_{domain} by 24 and the height of the domain tree respectively, and get the normalized distances $\hat{d}_{ip}$ and $\hat{d}_{domain}$ which are in the range $[0, 1]$. We define the network distance $d^{1,2}$ between H_1 and H_2 as:

$$d^{1,2} = \hat{d}_{ip}^{1,2} \times \hat{d}_{domain}^{1,2} \tag{1}$$

If H_1 and H_2 are in the same group, either from the IP tree or the domain tree, $d^{1,2}$ will be zero. Because transmissions between servers are also logged, we are able to evaluate the distance between two servers or between a server and a client.

4 Accelerate sharing

4.1 Replica generation

To accelerate sharing in a distributed storage system, it is very important to decide the number of data replications. In *Granary*, we use a simple thought that more replicas should be allocated for data that is more popular. Then the problem here is how we decide the replication number according to the popularity.

Researches on web caching reveal that the query frequency follows a Zipf-like distribution [10]. Researches on video-on-demand systems also show that replication distribution should follow the Zipf-like distribution [11]. Zipf-like distribution claims that the query frequency q_i of the ith popular object is proportional to $1/i^\alpha$, where α is a distribution-dependent constant. Based on Zipf-like distribution, we estimate the required number of replicas q_i for the ith popular file by

$$q_i = \frac{\beta M}{i^\alpha} \tag{2}$$

where M is the total query frequency of all files on all servers, and β is a constant configured according to the expected disk usage of the system. We assume that the servers are evenly loaded, so that M can be estimated by the total query frequency on a server (m) and the number of servers (N).

$$M = N \times m \tag{3}$$

Granary stores the logs of file access for a duration (e.g., 6 hours), and keeps a list of files sorted by request frequency. When a file is being requested, the server updates the list. Let r_i be the number of replicas of this file, if $r_i < q_i$, a new replica will be generated automatically.

However, the composition of the queries on a node in a distributed system is different from that in a single-server system. Since a client may have multiple servers to choose, the topological distribution of clients will influence the query distribution on a certain server.

For a certain client, we define a server as a *local server* if the network distance between them is below a predefined threshold (e.g. 0.5). To the opposite, we get a *remote server*. Then, requests sent to *local/remote server* can be named as *local/remote requests*. Although the server selection scheme described in Section 4.3 tends to choose a *local server*, a server may receive *remote requests* in two cases:

– *Local servers* are overloaded and the client have to choose a *remote server*;
– *Local servers* don't have a copy of the requested file.

Therefore, *remote requests* usually imply a misplacement of replicas and that we need to create a closer replica for these clients, while *local requests* better comply to the popularity distribution in the server's local domain.

For example, suppose Tom, a user at *CampusA*, has uploaded a video to *Granary*, and shares it to his friends. Then he notifies his friends about it by Email. It is possible that the friends in *CampusA* get Tom's message first and become the first client to download it. If the storage system finds out that this file is popular near *CampusA*, it will place all replicas near *CampusA*. Some days later, his friends in *CampusB*, which is far from *CampusA*, get Tom's Email and try to download the data. Because this file doesn't get more popular, based on the replication distribution strategies above, no more replicas will be generated. So there will never be replicas near *CampusB*, even though it's quite slow for the friends in *CampusB* to download from servers near *CampusA*.

In *Granary*, the problem is fixed by a mechanism called *Remote Boosting* (RB). We compare these two kinds of requests. If a file has more *remote requests* than *local requests*, even if its replica number is enough according to its popularity, the node will create a new replica for it. The replica placement scheme described in Section 4.2 will ensure that the new replica to be created near *CampusB*. After the new replica has been successfully created, the initial replica in *CampusA* will be deleted to accord with the file's popularity. This deletion does not actually remove the replica from the storage server. Instead, it is just removed from the file's replica list and marked as deleted (trashed) on the storage server. A trashed replica will be really deleted when the server needs to spare disk space for new replicas, but if the file is decided to replicate on this server again, the trashed replica (if survived) could be recycled so that the cost of replica propagation would be saved.

4.2 Replica propagation

The optimal placement of a new replica is to place it near users demanding it. We quantize the benefit of placing a replica of file f on a certain server a by the average transmission distance $\bar{D}(a)$.

$$\bar{D}(a) = \frac{\sum_j d^{j,a} \times q_j}{\sum_j q_j} \tag{4}$$

where $d^{j,a}$ is the network distance from client j to server a, and q_j indicates the request frequency from client j for file f. We will select the server with the minimal $\bar{D}(a)$ as the *destination server*, on which we make a new replica.

The data of new replicas is distributed in a pulling manner. Let's suppose an initiating server A has decided to propagate file f to *destination server* B. Firstly, server A adds the ID of B in the *replica list* of file f. In *Granary*, the replica list is stored in its DHT layer which ensures consistency and availability, and can be globally accessed. Then server A sends a message to B to notify about it. After that, server B downloads the file content from some server, which is not necessarily A, but is chosen by the server selection scheme described in the next section.

This method ensures the replicas to be propagated as soon as possible. Additionally, since the replica list is already updated, clients can start to download from server B even before the replica propagation is finished. In fact, they often complete the download almost simultaneously with the propagating process. This is because that the network speed from a client to a server is often slower than that between two servers in *Granary*. Apparently, the effect is the same as setting up a proxy server on B. It is just another benefit from our strategies.

If there is no enough space on *destination server* to store the new replica, some file on the node will be deleted to spare the storage space. We use the *least recently used* (LRU) scheme to decide which file to delete. However, it means a reduce of replicas for some file. To ensure data availability, a minimal number replicas (e.g. 2) is persisted for every file. If the node is unable to allocate enough disk space for the file content, it will remove itself from the replica list in the meta data thus cancels this replication. However, if such kind of things happens frequently, what we really need to do is put more servers in this area.

4.3 Selecting a proper server

In *Granary*, a client logs in to a server he knows first, which is called the *portal*. The *portal* can be selected from the server list stored on the client, which will be updated every time he logs in. With this design, client can get the latest *portal* list, and access the system even when portal list update is unavailable.

After accessing the selected *portal*, a list of available servers will be sent back to the client. It includes nodes with enough disk space when uploading. As for downloading, it is composed of those accommodating the file's replicas, which can be learned from the meta data stored in the DHT layer. After that, the client broadcasts a request message to all available servers. Thus each server will send back a reply message, including an estimated *weighted mean data rate* ($\bar{R}$) and the *network distance* between the server and the client.

The $\bar{R}$ is calculated from the logs recorded in the requesting client's cluster leaf in the IP tree or domain tree, as indicated below. In such formula, j is the index of the log; $size_j$ indicates the size of the file transmitted; and age_j shows how many hours it has been since the transmission finished; while $time_j$ describes the time the transmission takes.

$$\bar{R} = \frac{\sum_j size_j \times 0.5^{age_j}}{\sum_j time_j \times 0.5^{age_j}} \tag{5}$$

The $\bar{R}$ reflects the most recent transfer speed to and from the requesting client or other clients near it. We use it to prevent the convergence of client's selection. If every client simply chooses the closest server, it may happen that most clients converge to a small fraction of servers and thus overload them, while other servers are still idle [12]. The client picks half of the available servers with higher $\bar{R}$, from which it then selects the one with the shortest distance. Because an overloaded server concedes a low $\bar{R}$, it will be rejected in the first step. This server selection scheme is named as the *Hybrid* selection.

5 Evaluation

Granary has been deployed in 5 universities across China. It has attracted about 500 registered users to use it. Each user stores about 800 megabytes data in average in the system. The scale of the current system is still too small for a strategy evaluation, so we use two kinds of experiments to demonstrate the effectiveness of our replication schemes. The first one is a network simulation using logs from two heavy loaded sharing systems. And the second is an emulation carried out on PlanetLab.

5.1 Simulation results

In this section, two simulations were done for evaluating the effectiveness of the strategies used under heavy workload. The first one was carried using a transfer log from a FTP server mainly used for multimedia file sharing in our campus, and the other was finished with request logs from a large scale wide-area network video on demand(VoD) system [13]. Apparently, FTP is a traditional way of file sharing. As for VoD, it can be looked as a kind of real time sharing method. Both systems are used for multimedia sharing, which is also targeted by *Granary*. The details of these two workloads are shown in Table 1. We developed a simulation program that simulates a *Granary* system running in a bandwidth-limited wide-area environment. We ran simulations on a network topology generated by GT-ITM[1]. The network consisted of 9 transit domains, each of which owned 12 transit nodes in average. A transit node was connected with 1 stub domain which had 10 stub nodes commonly. Thus in total there were 108 transit nodes and 1080 stub nodes. The bandwidth of inter-domain links was 1Gbps and that of intra-domain links was 100Mbps. Each client got a bandwidth of 10Mbps. Finally, forty stub nodes were chosen randomly to be the storage servers, while the others would be assigned as client hosts.

We have evaluated several combinations of techniques in the simulations. Before introducing the experimental results, we firstly explain the terminology that would be used below.

- *Caching* – the caching scheme used by Pond [2] and PAST [3]. The client requests the nearest server for a file, which retrieves the file from the accommodating servers, sends it to the client and keeps it in its local cache.

[1] GT-ITM project, http://www-static.cc.gatech.edu/projects/gtitm/

Table 1. Characteristics of the simulation workloads

	FTP	VoD
Duration	136 days	30 days
Number of files	8,540	7,525
Average file size	56MB	125MB
Number of requests	97,239	3,021,401
Requests per hour	30	4,196
Size of downloaded data	12,573GB	264,562GB

Table 2. The *average transmission rate of downloads* (Avg. rate) and the *average traffic on network links* (Avg. traffic) in the simulations using the FTP and VoD workloads

	Avg. rate (byte/s)		Avg. traffic (MB)	
	FTP	VoD	FTP	VoD
Caching	773,588	634,381	13,220	191,400
Granary (C)	773,338	828,779	13,280	163,700
Granary (C + RB)	773,655	830,062	13,260	163,000
Granary (H)	773,728	838,942	13,250	162,200
Granary (H + RB)	774,172	840,365	13,220	161,400
Full mirror	900,269	892,462	5,997	120,700

– *Granary* – the replication scheme described in this paper and used in the *Granary* system. It may come with one or two of the following strategies:
  - *Closest* (C) – the client always choose the server with the shortest distance, using the network distance metric defined in Section 3.2.
  - *Hybrid* (H) – the client choose half of the available servers with higher *weighted mean data rate* ($\bar{R}$), from which it chooses the one with the shortest distance. It is explained in Section 4.3.
  - *Remote Boosting* (RB) – the server propagates extra replicas for files that are not very popular but have more remote requests than local requests. See Section 4.2 for details.

– *Full mirror* – all files have already been mirrored on every server. It gets the optimum performance at the cost of maximum disk usage.

Table 2 shows that with the VoD workload the average download rate under *Granary* (H + RB) is 32.5% higher than under *Caching*, while the improvement is not so obvious with the FTP workload. After further analysis, we found that in FTP workload the requests for a single file is evenly scattered over a long period of time, so that every file is requested at a rather low frequency. Cache and replication are usually ineffective when files are not frequently requested. This can explain why the result of FTP workload is not as good as that of the VoD's.

From Figure 1, we can find that *Granary* (H + RB) has reduced the fraction of downloads with transfer rate below 700KB/s from 20% to 10%, which is an obvious improvement of user experience.

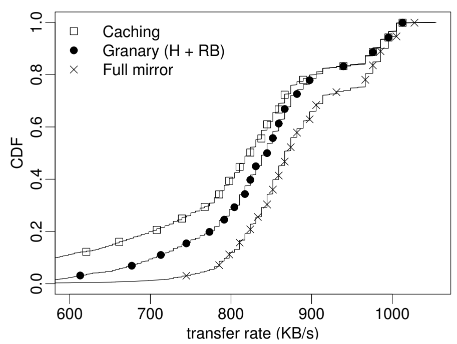

Fig. 1. The CDF of transmission rate of downloads (VoD workload)

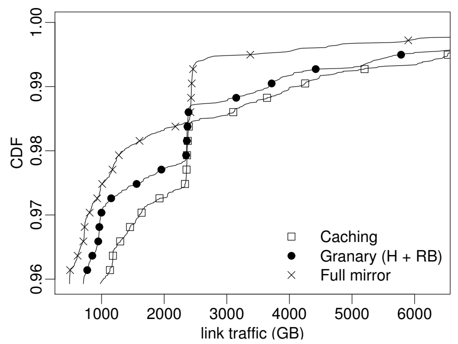

Fig. 2. The CDF of link traffic on the most critical links (VoD workload)

Then let's have a look at the bandwidth consumption. We recorded the total throughput on every network link that connects two hosts. Table 2 demonstrates that *Granary* replication schemes consume less network bandwidth than *Caching*. It is clear from Figure 2 that network traffic on the most critical links, which are the ones with the biggest throughput, has been notably reduced by *Granary*.

5.2 Experiment results on PlanetLab

PlanetLab[2] is an Internet-scale testbed for distributed systems. To test the performance of *Granary* in a wide-area network, we deployed *Granary* on 40 nodes of PlanetLab, which are located dispersively in different countries all over the world including China, Singapore, US, Poland, UK, Spain, Denmark and Germany, and carried out two experiments: the *REPLAY* experiment and the *SHARE* experiment.

The *REPLAY* experiment is based on a request log from the *Granary* system that has been running as a public service in our campus for a few months. The log includes 746 download requests for 448 different files, from 60 different IP addresses. We doubled the density of the requests and assigned them to client simulators deployed on 120 PlanetLab nodes. The client simulators replayed the requests, and recorded the transmission rate of every download.

The *SHARE* experiment is to simulate a typical scenario of sharing, that a user uploads a file and tells his friends all over the world to download it. We set up client simulators on 193 randomly selected PlanetLab nodes, and made them download a 5MB-sized file one by one.

Four replication configurations are tested. The *naive replication* places 2 replicas on random servers and keeps them unmoved; the *near replication* places replicas close to the requesters; the *popular replication* decides the number of replicas by the file's popularity; the *near and popular replication* is the combination of

[2] PlanetLab, http://www.planet-lab.org/

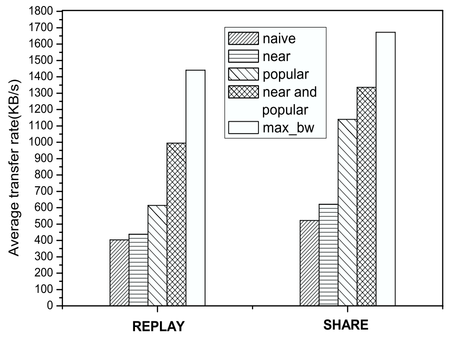

Fig. 3. Comparison of average transmission rate of downloads from experiments on PlanetLab

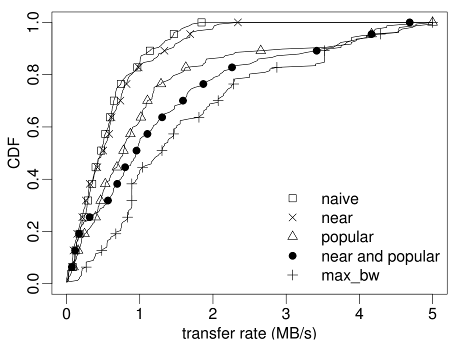

Fig. 4. CDF of transmission rate of downloads from the SHARE experiment on PlanetLab

the two, which is the default setting of *Granary*. In addition, we measure the maximal bandwidth of every client, and refer it as *max_bw* in the results.

The average transmission rate of downloads can be found in Figure 3. We can see that after applying *near and popular replication*, downloads have been accelerated by nearly 60% in the *REPLAY* experiment, and more than 100% in the *SHARE* experiment. The *SHARE* often performed better than the *REPLAY*, because a file is shared by more users in the *SHARE* than in the *REPLAY*.

Finally, we can have a further look to the *SHARE* experiment in Figure 4. The performance gap between different replication method is clearly shown. The CDF of *near and popular* is quite close to that of the *max_bw*, indicating it as a very optimized method.

6 Conclusion

More and more people store and share their files, including large files such as audio and video, on the Internet. There has been a lot of work improving specific file sharing systems such as BitTorrent, but improving the performance of file sharing on a distributed storage system still remains a challenge. In this paper, we introduced how we tried to accelerate the sharing of files on *Granary*, a distributed file storage system deployed in several campuses across China. *Granary* replicates popular files near to the users that request it for reducing its bandwidth consumption and improving the transmission rate of file downloads. Combined with the *Hybrid* server selection scheme and the *Remote Boosting* mechanism, our replication scheme performs notably better than conventional caching schemes in file sharing according to the experimental results.

References

1. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., Zhao, B.: Oceanstore: an architec-

ture for global-scale persistent storage. In: ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems, ACM Press (2000) 190–201

2. Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., Kubiatowicz, J.: Pond: the oceanstore prototype. In: FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, ACM Press (2003)
3. Rowstron, A., Druschel, P.: Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In: SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, ACM Press (2001) 188–201
4. Saito, Y., Karamanolis, C., Karlsson, M., Mahalingam, M.: Taming aggressive replication in the pangaea wide-area file system. In: OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation, Boston, Massachusetts, ACM Press (2002) 15–30
5. Hu, J., Li, M., Yu, H., Dong, H., Zheng, W.: Peerwindow: An efficient, heterogeneous, and autonomic node collection protocol. In: Proceedings of the 2005 International Conference on Parellel Processing (ICPP-05), IEEE Computer Society (2005)
6. Zheng, W., Hu, J., Li, M.: Granary: Architecture of object oriented internet storage service. In: CEC-EAST '04: Proceedings of the E-Commerce Technology for Dynamic E-Business, IEEE International Conference on (CEC-East'04), IEEE Computer Society (2004) 294–297
7. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with cfs. In: SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, ACM Press (2001) 202–215
8. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. In: Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability, Berlin/Heidelberg, Springer-Verlag (2001) 46–66
9. Andrews, M., Shepherd, B., Srinivasan, A., Winkler, P., Zane, F.: Clustering and server selection using passive monitoring. In: INFOCOM '02: Proceedings of the 21th Annual IEEE Conference on Computer Communications, IEEE Computer Society (2002) 1717–1725
10. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and zipf-like distributions: Evidence and implications. In: INFOCOM '99: Proceedings of the 18th Annual IEEE Conference on Computer Communications, IEEE Computer Society (1999) 126–134
11. Chervenak, A.L., Patterson, D.A., Katz, R.H.: Choosing the best storage system for video service. In: Multimedia '95: Proceedings of the third ACM international conference on Multimedia, San Francisco, California, United States, ACM Press (1995) 109–119
12. Mogul, J.C.: Emergent (mis)behavior vs. complex software systems. In: EuroSys '06, Proceedings of the 1st EuroSys Conference, Leuven, Belgium, ACM Press (2006) 293–304
13. Yu, H., Zheng, D., Zhao, B.Y., Zheng, W.: Understanding user behavior in large scale video-on-demand systems. In: EuroSys '06, Proceedings of the 1st EuroSys Conference, Leuven, Belgium, ACM Press (2006) 333–344