Qin A, Hu DM, Liu J *et al.* Fatman: Building reliable archival storage based on low-cost volunteer resources. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(2): 273–282 Mar. 2015. DOI 10.1007/s11390-015-1521-6

Fatman: Building Reliable Archival Storage Based on Low-Cost Volunteer Resources

An Qin (覃 安), Dian-Ming Hu (胡殿明), Jun Liu (刘 俊), Wen-Jun Yang (杨文君), and Dai Tan (谭 待)

Baidu Inc., Beijing 100193, China

E-mail: {qinan, hudianming, liujun01, yangwenjun, tandai02}@baidu.com

Received November 15, 2014; revised January 5, 2015.

Abstract We present Fatman, an enterprise-scale archival storage based on volunteer contribution resources from under-utilized web servers, usually deployed on thousands of nodes with spare storage capacity. Fatman is specifically designed for enhancing the utilization of existing storage resources and cutting down the hardware purchase cost. Two major concerned issues of the system design are maximizing the resource utilization of volunteer nodes without violating service level objectives (SLOs) and minimizing the cost without reducing the availability of archival system. Fatman has been widely deployed on tens of thousands of server nodes across several datacenters, providing more than 100 PB storage capacity and serving dozens of internal mass-data applications. The system realizes an efficient storage quota consolidation by strong isolation and budget limitation, to maximally support resource contribution without any degradation on host-level SLOs. It novelly improves data reliability by applying disk failure prediction to minish failure recovery cost, named fault-aware data management, dramatically reduces the mean time to repair (MTTR) by 76.3% and decreases file crash ratio by 35% on real-life product workload.

Keywords volunteer storage, failure prediction, failure recovery, reliability, archival storage

1 Introduction

Internet companies have collected billions of gigabytes of data on their storages during recent years, and have to face the growing of storage cost. These large sets of data that are not always processed by applications still spend the space of expensive storage resources on offline computational clusters. According to ESG (Enterprise Strategy Group) report①, 80% of the file data on storage systems is wasting high cost resources for their inactivation. These sets of data should be transferred to the archiving system. However, maintaining huge quantities of data still has to pay for extra machines, racks, and rents.

Many studies for building cost-saving archiving systems focus on compressing high-density data[1-2] and shifting cost to opening cloud storage providers like Amazon S3 or Windows Azure[3-4]. However, the main cost lies in the machine purchase and attached cash on datacenter infrastructure. They seldom think about how to make use of existing unutilized storage nodes to build the system. Erasure codes technique[5-6] is currently used in big-data storage since it can cut down almost half of storage volume without reliability loss, but it brings in a tenfold overhead in network bandwidth and disk I/O during the recovery process. The shifting cost on opening cloud storage cannot actually cut the cost. It limits the flexibility of company applications by restricted service interfaces of third-party providers.

On the other hand, the average disk utilization on existing clusters is always low. To cut down the cost, it becomes a trend to adopt compact hardware configuration in machine purchase, which redundantly attaches several disks to serve both CPU-bound and IO-bound applications at the same time. This compact configuration is beneficial for internet companies. Since internet business generally consists of offline backend for IO-bound analysis and online frontend for CPU-bound

Regular Paper
Special Section on Applications and Industry

① Asaro T. File system archiving. http://www.emcemea.com/materials/FileSystemAssess/ESG_FSA_Whitepaper.pdf, Nov. 2014.

query processing, mixed deployment on compact configured hardware can reduce much overhead when transmitting tons of result data from offline clusters to online ones. However, compact configuration introduces low utilization when the hardware requirement of two kinds of applications does not match mutually. Meanwhile, legacy servers with common hardware configuration do not fully consume all hardware capacity. Therefore, the spare capacity can be contributed as volunteer resources. From our statistics on the datacenters of the biggest internet search service provider in China, there exists more than 100 PB free storage space on tens of thousands of frontend servers, and in most of them, the utilization on both space and IO is lower than 40% over years.

In this paper, we present Fatman, a novel design for enterprise-scale archiving storage built on volunteer nodes, which makes use of the idle storage contribution to implement PB-level low-cost reliable storage. The volunteer node can be any one of servers from search backend or frontend, which makes agreement of resource sharing and gets protection from malicious behaviours. Our contributions are as follows.

• We investigate the challenges of archiving systems building on volunteer storage. We address the basic isolation requirement, the features of various quality of service (QoS) of storage medium, and the complexity of data reliability. (Section 2)

• We present the system architecture and the lightweight isolation mechanism to implement budget-based resource limitation. (Subsection 3.2)

• We outline the rules to place the data replicas within resource limitation, and show how to take advantage of medium heterogeneity without losing reliability. (Subsection 3.3)

• We demonstrate pre-scheduled data recovery based on failure prediction of disk medium, which efficiently avoids the hardware failure. (Subsection 3.4)

• We finally demonstrate by experiments that Fatman can ensure the reliability together with cost efficiency. (Section 4)

2 Challenge

The major barriers ahead of impeding broader applications of volunteer storage systems lie in three aspects.

• Resources isolation and limitation, is to guarantee no performance influence on host applications. Like TFS[7], contributory application is transparently running as parasite daemon in volunteer nodes, sharing the host's CPU, memory, disk, network, and other local resources. The hybrid-deployed contributory application will cause heavy performance degradation as more storage is allocated[8] when no isolation and limitation are enforced in resource usages.

• Heterogeneous storage medium, usually contributed to volunteer storage, has influence on hardware performance, reliability, and cost[9]. It is critical for Fatman and other volunteer storages to accomplish isolation and abstraction for slow, failed or possibly-fail storage medium and to minimize the data failure repair overhead for the purpose of avoiding the host's service level objective (SLO) violation. Especially for those replica recovery with erasure codes[5-6], data recovery is expensive under all kinds of medium fault.

• Reliability is complicated and tricky. Besides heterogeneous resource failure, contributory applications may be deliberately killed at any time for resource withdrawn by volunteer nodes. Activated replication recovering cannot be quickly processed because of resource limitation and non-priority access. To smooth resource utilization without reducing availability, assumptions are pre-required in some systems to determine when and how data will be recovered[7]. But these assumptions are not always true in general cases[9-10].

3 Design and Implementation

3.1 Overview

Fatman adopts master-slave architecture: metadata is maintained by the master and file data is stored via *datanode*, which acts as the contributory service, residing in volunteer nodes and trying to share local resources (see Fig.1). For scalability, there are several metaservers in the master assisting with metadata management.

Resource isolation and limitation are implemented in *datanode* to monitor the usage of CPU, memory, disk, and network. Hardware health parameters are also collected to be used for failure model training and failure prediction, which would provide hints for scheduling on data recovery and power efficiency.

3.2 Resource Availability

To enforce network bandwidth within given limitation (say b MB/s), the network is scheduled based on budget. Each second will be assigned with a budge of b MB for the total sending buffer size on RPC channel. During a second, each RPC request will subtract part of budget according to its sending package size. When

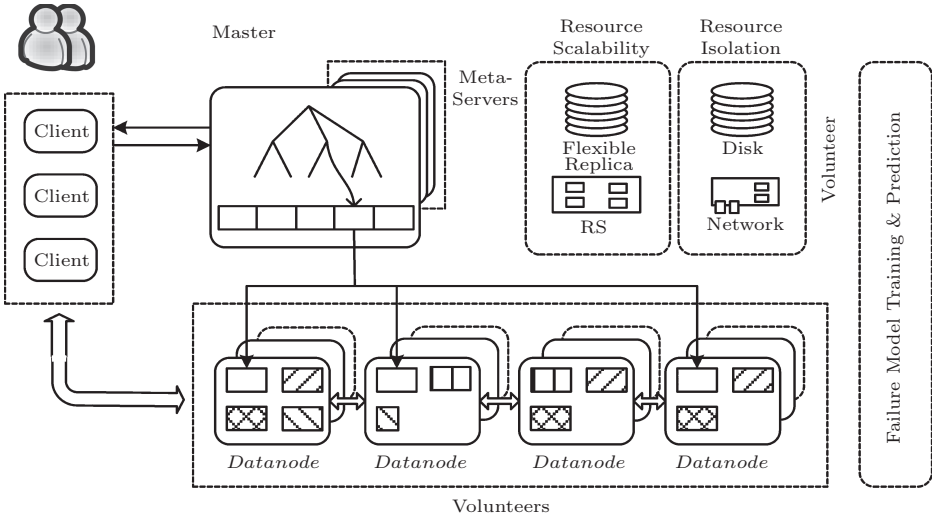

Fig.1. Overall architecture.

the budget is exhausted, the next RPC request will be blocked until new budget is assigned in the next second. Sometimes, it needs to fine-tune the parameters to avoid network peaks hurt the host's local applications.

Budget-based scheduling is also used to control disk IO bandwidth. But for multiple block devices, all the block devices are sharing the total budget. Currently, storage capacity is only isolated via physical disks, and is returned via background garbage collector.

CPU and memory are controlled elastically by cooperating with the *self-manager*, which checks whether any over-consumption of resources has occurred, and will execute suicide to clean malignant affect.

Since the *self-manager* is designed as a stand-alone daemon, it can watch all contributory daemons and audit their resource consumption objectively. Different from the hypervisor of virtual machines[11], it is more lightweight and can be launched or destroyed quickly. Also different from kernel-level container (i.e., cgroup)[12], the *self-manager* only controls CPU and memory. On the other hand, we do not adopt virtual machine or kernel-level container because almost all volunteer nodes are running online services, which does not allow for deploying new system softwares or shutdowning to upgrade kernel version to support new features.

3.3 Replica Placement

Data in Fatman are classified as hot data with three replicas and cold data encoded via Reed-Solomon (RS) algorithm[5-6]. As in [13], a big file will be separated into 256 MB-size data blocks, stored in distributed *datanode*. Each data block may have replications, or may be encoded together with several parity blocks. The archival data are hot when they are just written in the system or accessed recently. With the classification of hot data and cold data, we can simply know which data are inactive and can save storage capacity by applying algorithms with high compression rates.

The placement aims at reaching the balance between low cost and high availability. For three-replica hot data, one block replica is placed in high-quality storage media, while the other two mirrors are in cheaper media (we will try to place one of them in neighboring datacenter). For RS-encoded cold data, data blocks are tried to be placed in a cheaper but better-performance medium, while parity blocks are placed in a high-available medium. The storage mediums are ranked by their service quality based on their hardware parameters, service time, repair history, etc. Therefore, accessing cold data will achieve better performance in most cases without data block crashes.

The placement of cold data helps to save net bandwidth on data recovering. If data block crash happens, the recovery of cold data has to access all the data blocks and parts of the parity blocks, which would cause heavy IO consumption. To achieve better recovering performance, an optional solution is to split cold data block into slices and disperse them across several nodes. Fatman adopts RS(10, 4) to encode the data slices, where RS(n, k) is defined to represent that one data is split to n data blocks with k parity blocks. Fig.2 shows an placement example of RS(3, 1).

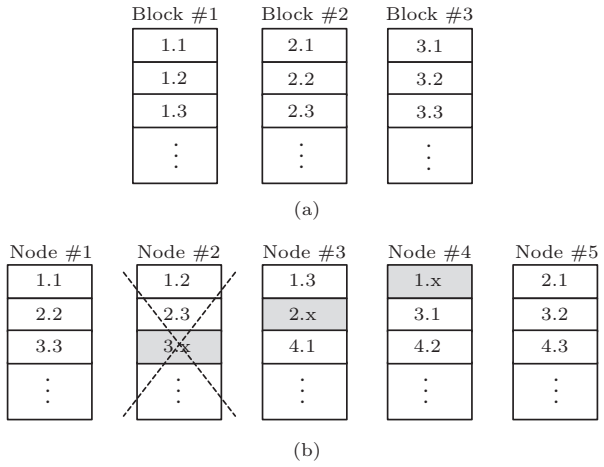

Fig.2. RS-encoded file split and placement. (a) File block and slice. (b) Block crash and recovery.

Each slice owns 10 segments for data and 4 segments for parities. For un-failure data slice access, the client only needs to read 10 data segments. However, when one needs to tolerate t failure block(s) ($t < 4$), since each slice will have at most 4 failure segments, he/she will read $10 - t$ data segments and t parity segment(s) back in stream to recover the slice on crashed blocks. To recover one crashed data segment, other data segments will be prefetched from *datanode*, and these segments can be used again to assemble the whole block. For hot data, the read on failure data block will automatically switch to another replica.

3.4 Fault Awareness

Because of the resource limitation, data recovering needs to be scheduled intelligently in advance to reduce the mean time to repair (MTTR) of the failure file data. The pre-scheduling strategy is cooperated with hardware failure prediction. Usually, the prediction mechanism notifies the scheduler to prepare data transmission several days in advance, and then the scheduler makes use of the idle time and activates replication recovering.

To the best of our knowledge, the accurate detection of storage failure can help the system to optimize the MTTR[14], since the speed of repair after a failure determines the reliability for a specific storage system[15]. However, the classical erasure codes relied on by Fatman, such as RS codes, are suboptimal for distributed environments because RS(10, 4) recovery requires to transfer 10 blocks and recreate the original 10 data blocks even if a single block is lost[16]. Hence, the recovery process suffers from a tenfold overhead in repair bandwidth and disk I/O. To be more efficient, locally repairable codes (LRCs) have been introduced in [1-2, 17], which gain 50% disk I/O and network reduction at the cost of 14% more storage.

As reported in [18], 78% of the hardware replacements in modern datacenters are caused by hard disk failures. Therefore, Fatman applies disk failure prediction to improve the MTTR, and our result shows a better performance than LRCs without any more storage overhead. In our previous work[19], our model can predict 84.8% of the failures at least 24 hours ahead, which will provide with enough time margin for data migration from possibly-fail drives.

Once Fatman receives the alerts of incoming disk failures, it can pre-schedule the recovery process as soon as possible. Therefore the data reconstruction time after the failures will be reduced, especially when heavy RS decoding is needed under computing resource limitations. The benefits of this mechanism to replica robustness lie in two aspects. On one hand, as [20] has confirmed, a system that can predict failures sufficiently ahead of time would be able to extend the mean time to failures (MTTF) and shorten the MTTR evidently. On the other hand, it gives an opportunity for Fatman to migrate and recover data when the resource budget of the target node is unstrained, which means that higher bandwidth (network and disk) and more CPU/memory can be used while side effects to basic performance are under control. We will show the effectiveness of the failure prediction method in Subsection 4.3.

4 Evaluation

The goals of our evaluation here are 1) to illustrate that volunteer resources are available for Fatman without any resident influence; 2) to show the performance based on volunteer environment with specific resource limitation and enforcement; and 3) to demonstrate the reliability impact of fault-aware data management and scheduling on the MTTR and real-life product workload.

4.1 Resource Availability

We measure the resource consumption on one of the real online volunteer nodes, which is configured with 8-core 2.4 GHz Xeon processors, 32 GB of memory, 12 1 TB disks, and a 1 Gbps full-duplex Ethernet connection. Fatman *datanode* and *self-manager* services are deployed within the same Linux account in the volunteer node. In Fig.3, the workload fluctuation (line *host*) is simulated from a real local host application to measure the impact on contributory services (*datanode*)

of their resource consumption (line *contributory*). The measurement can be easily conducted via system tools or commands (e.g., *top* or *ps*).

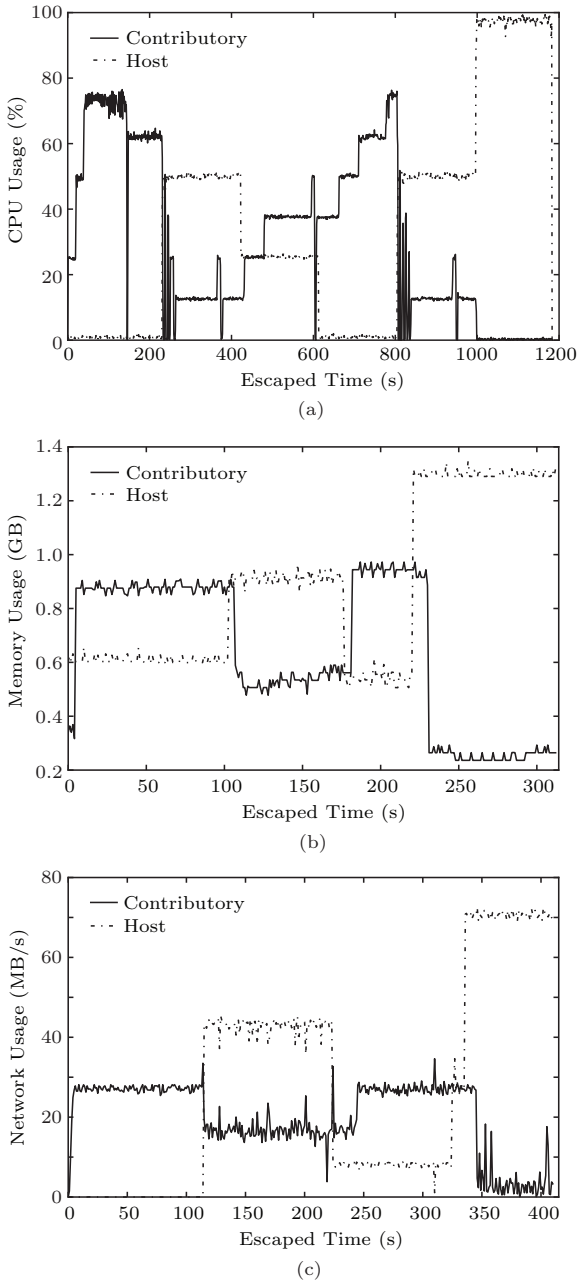

Fig.3. Elastic resource usage for dynamically changed workload. (a) Elastic CPU usage. (b) Elastic memory usage. (c) Elastic network usage.

CPU is a flexible resource and can be easily scheduled without killing processes. Usually, *datanode* keeps the core consumption under specific threshold (say 60%) when the host workload is not heavy. When host workload increases and challenges more CPU resource, *datanode*s will gradually return the core. In Fig.3(a), we simulate the workload changing as the fluctuation of the dot-dash line. It can be seen from the figure that the *datanode* withdraws the resource consumption to ensure the host's resource provision. When the host workload continues to increase to the upper limit, the *datanode* would be totally killed to release all the resources, which can be seen from the black line at 1 000-second point.

Differently, the memory is recycled via killing contributory services (Fig.3(b), 1 GB limited). We use *kill*, instead of *shut down*, because the execution is quicker and simpler to release resources. The *self-manager* can be optionally restarted after suicide, which is configured in Linux *cron* service.

The budget-based network scheduling can achieve a relatively stable utilization of contributory services. In our experiment in which the network bandwidth is limited to 30 MB/s, we can see from Fig.3(c) that *datanode*s will keep this level unless the resource challenge arrives at the threshold from host workload. Similarly, the *self-manager* detects the resource shortage and informs *datanode*s to narrow down the bandwidth budget whenever necessary.

As the disk capacity resource is recycled via garbage collector in background, the effect of resource release will not be instant. But, it has no influence on our real workload, because the disk utilization of real workload is almost stable and seldom approaches the threshold. Additionally, the physical isolation based on devices is common in our real-world deployment.

4.2 Performance

We measured the performance on a Fatman cluster consisting of one master, two master replicas, 24 datanode servers, and 32 clients. Hardware configuration of each datanode server is the same as the real volunteer node. We prepare both hot and cold data in our experiment: hot data has three replicas for each data block, and cold data is encoded with RS(10, 4). Note that this configuration was set up for the ease of testing, while general clusters have several thousands of datanode servers and thousands of clients.

4.2.1 Reads

N clients read simultaneously from the system. Each client reads a randomly selected 256 MB block size from a 500 GB file set without replication crash. This is repeated 10 times to simulate the big-data flowing

operation like the real case. Fig.4(a) illustrates the aggregated read throughput with the increase of the client number. As for hot data with three replicas, read operation can easily reach 120 MB/s peak limit of 1 Gbps switch, and 14 MB/s for each client on average. But for RS-encoded cold data, the peak throughput fluctuates at around 85 MB/s (14 MB/s for each client). Because the RS-decoding is processed in stream from data segments across tens of *datanode* which swaps out some CPU time from networking, the curve in Fig.4(a) cannot reach the bandwidth of hot data when increasing the client's workload.

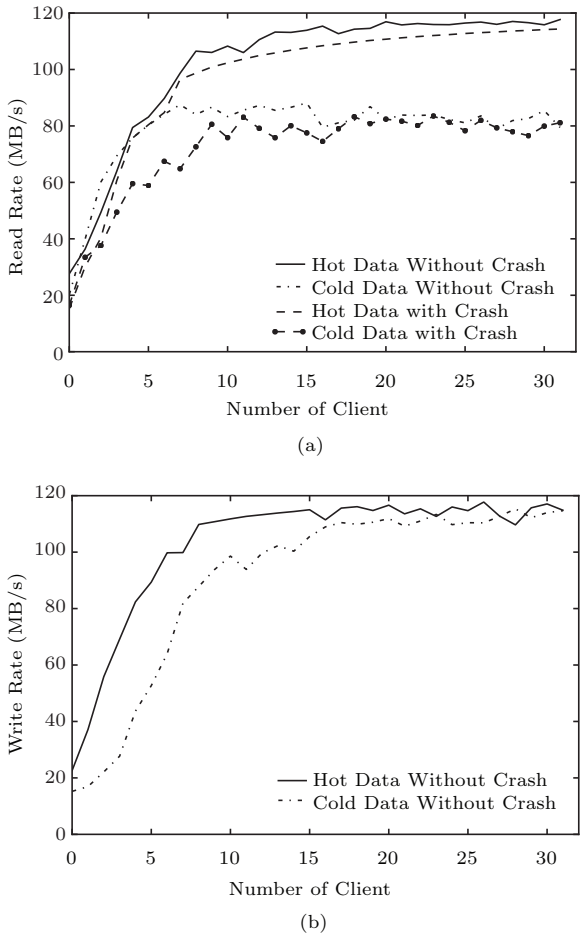

Fig.4. Read/write throughput. (a) Reads. (b) Writes.

4.2.2 Writes

N clients write simultaneously to N distinct files. Each client writes 1 GB of data to a new file in a series of 2 MB writes. The write throughput is shown in Fig.4(b). Averagely, the write for both of hot and cold data can reach the peek speed 14 MB/s per client. But the throughput for cold data cannot keep up with the corresponding value of hot data at full workload.

The reason of low throughput at full workload is that RS computation slows down the networking. Since computation resource is strictly limited and the current implementation has to calculate the parity segments per ten data segments, there exists an idle scope of networking flow reducing the throughput. But the effect of the idle scope can disappear gradually after $N > 16$ (see Fig.4(b)).

4.2.3 Reads on Failure Replication

Fig.4(a) also shows the performance of reads on those files with some crashed replicas. The experiment simulates the replica crash via randomly deleting replication without crashing the whole file. For hot data, we randomly delete one or two replica(s), and for RS-encoded cold data, one or two block(s) is/are selected to be deleted. The missing reads can be used to measure the overhead of data recovering when the client tries to read back the correct data from the system.

As shown in Fig.4(a), the missing reads have minor influence on client throughput for hot data, which is decreased by 0.4%. But for cold data, it is 29.7% averagely. For hot data, the client usually selects a closer and lower-workload replica to execute data read. If the replica is crashed or missing, the performance cost of the client is only the switch to the next replica. Therefore, the crash-replica read throughput of hot data is almost equal to no-crash ones. But for cold data, crashed replica would cause multiple additional read operations of parity segments to recover the data segments. Along with the computation cost, that will decrease the throughput.

4.2.4 Data Recovering

Table 1 lists the recovering performance of a single replica in Fatman (for RS-encoded cold data, each block only has one replica). For hot data, it is determined by networking and disk IO. While for cold data, RS-based decoding also needs to consider CPU and memory, which may result in some performance loss during recovering. Since one crash/missing RS block will fetch a tenfold dataset, it needs to open ten connections averagely and read geographically from block files. Under the 40 MB/s bandwidth, it needs 82.8 seconds to recover one block file.

Table 1. Recovering Performance of Single Block

Type	Block Size (MB)	Trans. Size (MB)	Opened Connections	Network Limit (MB/s)	Consumed Time (s)
Three-replica	256	256	1	40	7.25
RS-encoded	256	256×10	10	40	82.80

4.3 Fault Awareness

As discussed in Subsection 3.4, Fatman implements a fault-aware recovering mechanism by predicting incoming disk failures, which can reduce 76.3% of the MTTR for RS recovery. For those possibly-fail disks, Fatman not only pre-schedules the recovery processes for the sake of resource limitation to improve the repair efficiency, but also cuts down the reconstruction cost apparently, since the system only needs to migrate the data away from the possibly-fail drives, which consumes only 1/10 repair time of RS recovery time. The actual statistics shows that 1 384 of the predictions were made 24 hours ahead of 1 632 failures, accounting for 84.8% of all the failed disks (shown by the cumulative distribution function in Fig.5). Therefore, the MTTR of RS should be reduced by at least $(1-1/10)\times 84.8\% = 76.3\%$ if the latent failures can be predicted one day ahead, which will save the computing resources for RS decoding in the meantime.

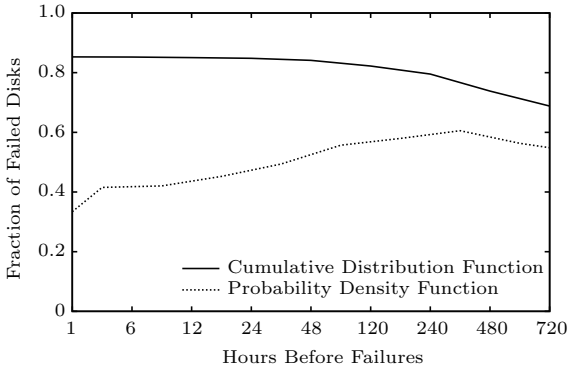

Fig.5. Distribution of failed disks under different time margins.

4.4 Reliability Improvement

We specify some status to represent the file block changing. For three-replication hot data, f_1 means the client hits one failure replication (including missing replica), f_2 means two hits, and f_3 means the file has crashed since at least one block in the file has been completely missed. Fig.6 shows the three-month statistics from log files of our daily-run process routine, which is real-life product workload of reading 1T data randomly from our testbed. Comparing the no-prediction and the prediction of hot data, we can find the failure prediction situations can help us cut down 77% of the crash replications (f_3) and 68% of two-replica failures (f_2). In Fig.6, the number of one-hit prediction is higher than the no-prediction one, because two-failure-replication block will be recovered first, which increases the number of one-failure ones.

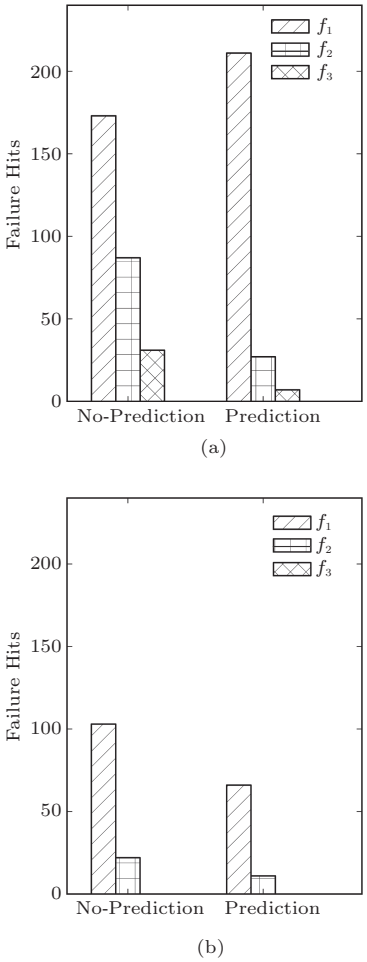

Fig.6. Failure hits of read statistics. (a) Three-replica data. (b) RS-encoded data.

For RS-encoded cold data, f_2 is specified to the status in which the failure replication number is between two and k, where k is the maximally tolerant crash number in RS. In Fatman's RS code, each ten units of data segments will create four units of parities, and thus here k is 4. Therefore, total crashed replica on f_1 is less than

that of hot data, and the benefit of failure prediction can achieve 35% for f_1 and 49% for f_2. But for f_3, both of them are zero, which means that no file is crashed or missing in our experiments.

5 Related Work

Volunteer computing aims at enhancing the resource utilization without local performance loss. Currently, prevalent volunteer systems or products are built based on computational resources like CPU and memory, since this category of runtime resources is relatively easier in control[21]. It introduces more challenges of volunteer storage to solve the efficient utilization of heterogeneous host resources with varying capabilities and the instability of high optional participation rate, which are two of basic requirements in volunteer computing[22].

Volunteer storage would cause SLO violation without efficient isolation during utilizing volunteer resources. VM-based isolation can achieve strong limitation but also introduces heavy overhead, while kernel-level container has not been ready for networking and disk IO[11-12]. Some still use the traditional audit-control method in application level, while it is coarse-grained and insensitive[7,21]. However, the application-level control can be pervasive for no resident requirements with existing applications.

The reliability of volunteer storage is more tricky, challenged by fault prevalence from the instability of volunteer resource provision or heterogeneous resource failure. As we have shown in Section 4, hard drive failure prediction can help maintain the reliability on storage system effectively. Merely relying on the Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T.) standard that most hard drive vendors will follow can predict at least 50 percent of future failures[20]. In order to further enhance the prediction accuracy, several machine learning techniques have been proposed using the S.M.A.R.T.-based feature sets[23-25]. In our previous work[19], we also compared the performance of backpropagation neural network and SVM.

Storage system has long been tried to build more reliable systems via disk mirroring[26] and RAID[27]. Erasure codes[5-6,28] can improve reliability with low storage capacity but the overhead of using such erasure codes will likely reduce system performance.

The integration of failure prediction and erasure codes is seldom reported in recent articles, especially applying prediction to pre-schedule decoding. Previous work mainly focuses on self-monitoring detection, but does not consider hardware-aware tolerance design. The work in this paper is based on [29], but presents more details about replica placement on RS-coded data slice to achieve the lowest overhead for data recovery and the summary of volunteer environment in real product workload. Meanwhile, this paper reaches further performance enhancement than [29] during recovery via sharing common blocks between replicas.

6 Conclusions

Volunteer resource contribution is one of the obvious ways to achieve high scalability and low investment for building large-scale archival systems. However, traditional volunteer storage is hard to be popularized, which lies in the concerned influence on host applications against the reliable service provision. This paper presented Fatman, a novel storage system design on volunteer contribution resources to build a reliable enterprise-scale archiving storage system.

1) Fatman summarizes the challenges and the requirements of archiving system building on volunteer storage, and outlines the key designs focusing on resource availability, data placement, and fault awareness.

2) To guarantee the resource availability, Fatman is implemented with efficient budget-based resource limitation and can transparently share unutilized resources in full read/write throughput, without any degradation on host-level SLOs.

3) By fault-aware scheduler predicting potential hardware failure and perceiving QoS, Fatman can execute data scheduling and quality-aware data allocation in advance, achieving high availability and reliability.

4) The experiment result illustrates that failure replication ratio has been reduced by 68% for hot data and 35% for cold data at least and MTTR has reduced by 76.3%.

In reality, Fatman has been deployed on tens of thousands of server nodes across several datacenters, providing more than 100 PB storage capacity and saving millions of dollars for company. The storage capacity of Fatman has served for dozens of business applications, especially for those valuable datasets with periodic updates like webpage processing and user business logging.

References

[1] Sathiamoorthy M, Asteris M, Papailiopoulos D S, Dimakis A G, Vadali R, Chen S, Borthakur D. XORing elephants:

Novel erasure codes for big data. In *Proc. the 39th VLDB*, Aug. 2013, pp.325–336.

[2] Huang C, Simitci H, Xu Y, Ogus A, Calder B, Gopalan P, Li J, Yekhanin S. Erasure coding in windows Azure storage. In *Proc. USENIX ATC*, Jun. 2012.

[3] Vrable M, Savage S, Voelker G M. Cumulus: Filesystem backup to the cloud. In *Proc. the 7th USENIX Conf. File and Storage Technologies*, Feb. 2009, pp.225–238.

[4] Vrable M, Savage S, Voelker G M. BlueSky: A cloud-backed file system for the enterprise. In *Proc. the 10th USENIX Conf. File and Storage Technologies*, Feb. 2012, pp.19:1–19:14.

[5] Reed I S, Solomon G. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 1960, 8(2): 300–304.

[6] Khan O, Burns A, Plank J, Pierce W, Huang C. Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads. In *Proc. the 10th USENIX Conf. File and Storage Technologies*, Feb. 2012, pp.20:1–20:14.

[7] Cipar J, Corner M D, Berger E D. TFS: A transparent file system for contributory storage. In *Proc. the 5th USENIX Conf. File and Storage Technologies*, Feb. 2007, pp.215–229.

[8] McKusick M K, Joy W N, Leffler S J, Fabry R S. A fast file system for UNIX. *ACM Trans. Comput. Syst.*, 1984, 2(3): 181–197.

[9] Hoelzle U, Barroso L A. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines (1st edition). Morgan and Claypool Publishers, 2009.

[10] Schroeder B, Gibson G A. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proc. the 5th USENIX Conf. File and Storage Technologies*, Feb. 2007, pp.1:1–1:16.

[11] Barham P, Dragovic B, Fraser K, Hand S, Harris T L, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. In *Proc. the 19th SOSP*, Oct. 2003, pp.164–177.

[12] Soltesz S, Pötzl H, Fiuczynski M E, Bavier A C, Peterson L L. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proc. the 2nd EuroSys*, Mar. 2007, pp.275–287.

[13] Ghemawat S, Gobioff H, Leung S T. The Google file system. In *Proc. the 19th SOSP*, Oct. 2003, pp.29–43.

[14] Jiang W, Hu C, Zhou Y, Kanevsky A. Are disks the dominant contributor for storage failures? A comprehensive study of storage subsystem failure characteristics. In *Proc. the 6th USENIX Conf. File and Storage Technologies*, Feb. 2008, pp.111–125.

[15] Xin Q, Schwarz T J E, Miller E L. Disk infant mortality in large storage systems. In *Proc. the 13th MASCOTS*, Sept. 2005, pp.125–134.

[16] Rodrigues R, Liskov B. High availability in DHTS: Erasure coding vs. replication. In *Proc. the 4th IPTPS*, Feb. 2005, pp.226–239.

[17] Tamo I, Papailiopoulos D S, Dimakis A G. Optimal locally repairable codes and connections to matroid theory. In *Proc. CoRR*, Jan. 2013.

[18] Vishwanath K V, Nagappan N. Characterizing cloud computing hardware reliability. In *Proc. the 1st SoCC*, Jun. 2010, pp.193–204.

[19] Zhu B, Wang G, Liu X, Hu D, Lin S, Ma J. Proactive drive failure prediction for large scale storage systems. In *Proc. the 29th MSST*, Jun. 2013.

[20] Paris J, Schwarz T J E, Long D. Evaluating the reliability of storage systems. Technical Report, UH-CS-06-08, Department of Computer Science, University of Houston, 2006.

[21] Larson S M, Snow C D, Shirts M, Pande V S. Folding@home and Genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *arXiv: 0901.0866*, 2009. http://arxiv.org/abs/0901.0866, Jan. 2015.

[22] Durrani M N, Shamsi J A. Volunteer computing: Requirements, challenges, and solutions. *J. Network and Computer Applications*, 2014, 39: 369–380.

[23] Hamerly G, Elkan C. Bayesian approaches to failure prediction for disk drives. In *Proc. the 18th ICML*, Jun. 2001, pp.202–209.

[24] Hughes G F, Murray J F, Kreutz-Delgado K, Elkan C. Improved disk-drive failure warnings. *IEEE Transactions on Reliability*, 2002, 51(3): 350–357.

[25] Murray J F, Hughes G F, Kreutz-Delgado K. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research*, 2005, 6: 783–816.

[26] Bitton D, Gray J. Disk shadowing. In *Proc. the 14th VLDB*, Aug. 1988, pp.331–338.

[27] Chen P M, Lee E L, Gibson G A, Katz R H, Patterson D A. RAID: High-performance, reliable secondary storage. *ACM Comput. Surv.*, 1994, 26(2): 145–185.

[28] Plank J S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software — Practice & Experience*, 1997, 27(9): 995–1012.

[29] Qin A, Hu D, Liu J, Yang W, Tan D. Fatman: Cost-saving and reliable archival storage based on volunteer resources. In *Proc. the 40th VLDB*, Sept. 2014, pp.1748–1753.

An Qin works as a senior software engineer at Baidu and leads the team of storage infrastructure under the search engine system. His main areas of interest include large-scale distributed systems and high performance computing. He received his Ph.D. degree in computer science from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2010.

Dian-Ming Hu works as a senior software engineer at Baidu in charge of hardware failure, power and resource optimal management. His research interest focuses on data center architecture design and management techniques. He leads the hardware data mining team, initiating the intelligent IDC design and providing uniform hardware-aware service for Baidu infrastructure. He received his M.S. degree in software engineering from University of Science and Technology of China, Hefei, in 2011.

Jun Liu is a senior operation engineer at Baidu and leads the infrastructure operational team. His current research focuses on resource efficient cloud computing, automation and intelligent operation platform, distributed storage systems and infrastructure of search engine. He received his M.S. degree in biomedical engineering from Zhejiang University, Hangzhou, in 2007.

Wen-Jun Yang got his M.S. degree in computer architecture from Tsinghua University, Beijing, in 2013. He is currently working at Baidu as a senior software engineer. His research interest mainly focuses on distributed system management using machine learning/data mining techniques.

Dai Tan is currently a senior software architect in the Search Service Group of Baidu. His main fields of research include information retrieval, distributed systems and high performance computing. He is the leader of various projects of Baidu's crawling and indexing systems, along with the fundamental distributed computing and storage infrastructure. He received his B.S. degree in computer science from Huazhong University of Science and Technology, Wuhan, in 2008.