# Fatman: Cost-saving and reliable archival storage based on volunteer resources

An Qin, Dianming Hu, Jun Liu, Wenjun Yang, Dai Tan
Baidu, Inc
{qinan, hudianming, liujun01, yangwenjun, tandai02}@baidu.com

## ABSTRACT

We present Fatman, an enterprise-scale archival storage based on volunteer contribution resources from underutilized 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, provided more than 100PB storage capacity and served dozens of internal mass-data applications. The system realizes an efficient storage quota consolidation by strong isolation and budget limitation, to maximally support resources contribution without any degradation on host-level SLOs. It firstly improves data reliability by applying disk failure prediction to minish failure recovery cost, named fault-aware data management, dramatically reduces the MTTR by 76.3% and decreases file crash ratio by 35% on real-life product workload.

## 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 set of data are not always processed by applications but still spend the space of expensive storage resources on offline computational clusters. According to ESG report [1], 80% of file data on storage systems is wasting high cost resources for their inactivation. These set of data should be transferred to archiving system. However, saving huge number of data still has to be paid investment on extra machines, racks, and rents.

Many studies for building cost-saving archiving system focus on high-density data compression [21][11] or shifting cost to opening cloud storage provider like Amazon S3 or Windows Azure [27] [26]. They seldom think about how to make use of existing unutilized storage nodes to build the system, because the mainly cost lies in the machine purchase and attached cash on datacenter infrastructure.

On the other hand, the disk utilization on existing clusters are always low averagely. 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. Because internet business generally consists of offline backend for IO-bound mining or analysis and online frontend for CPU-bound query processing, mixed deployment on compact configured hardware can save much overhead when transmitting tons of result data from offline clusters to online ones. However, compact configuration will introduce low utilization when the hardware requirement of two kinds of applications does not match mutually. Another reason results in lower utilization is that legacy servers with common hardware configuration have not fully consumed all hardware capacity and these 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 240PB 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 server from search backend or frontend, which makes agreement of resource sharing and gets protection from malicious behaviours. Our contribution is as follows:

- We investigate the challenges of archiving system building on volunteer storage, addressing the basic isolation requirement, the features of various Quality of Service (QoS) of storage medium, and complexity of data reliability. (Section 2)
- We present the system architecture and lightweight isolation mechanism to implement budget-based resource limitation. (Section 3.2)
- We outline the rules to place the data replicas within resource limitation, and show how to take advantage of medium heterogeneity while not losing reliability. (Section 3.3)
- We demonstrate pre-scheduled data recovery based on


*Proceedings of the VLDB Endowment,* Vol. 7, No. 13

failure prediction of disk medium, which efficiently avoiding the hardware failure. (Section 3.4)

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

# 2. CHALLENGE

The major barriers impeding broader application of volunteer storage systems lies in three aspects:

- Resources isolation and limitation, to guarantee no performance influence on host applications. Like TFS [5], contributory application is transparently running as parasite daemon in volunteer nodes, sharing host's CPU, memory, disk, network and other local resources. The mixed-deploy contributory application will cause heavy performance degradation as more storage is allocated [16], if 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 [10]. It is critical for Fatman and other volunteer storages to accomplish the isolation and abstraction for slow, failed or will-fail storage medium and to minimize the data failure repair overhead for avoiding the host's SLO violation. Especially for those replica recovery with erasure codes [19][14], data recovery is expensive against all kinds of medium fault.
- Reliability is complicated and tricky. Besides heterogeneous resource failure, contributory applications may deliberately be killed at any time for resource withdrawn by volunteer node. Activated replication recovering can not be quickly processed because of resource limitation and non-priority access. To smooth resource utilization without losing availability, assumptions are pre-required in some systems to determine when data can be recovered and how to be recovered [5]. But the assumptions are not always true in general case [22][10].

# 3. DESIGN AND IMPLEMENTATION

## 3.1 Overview

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

Resource isolation and limitation is implemented in *datanode* to monitoring the usage of CPU, memory, disk and network. Hardware health parameters are also collected to failure model training and prediction, which result in-turn provides hints for scheduling on data recovery or power efficiency.

## 3.2 Resource Availability

To enforce network bandwidth within given limitation level (say $b$ MB/s), network is scheduled based on budget. Each second will be assigned a budge of $b$ MB for the total sending buffer size on RPC channel. During this second, each

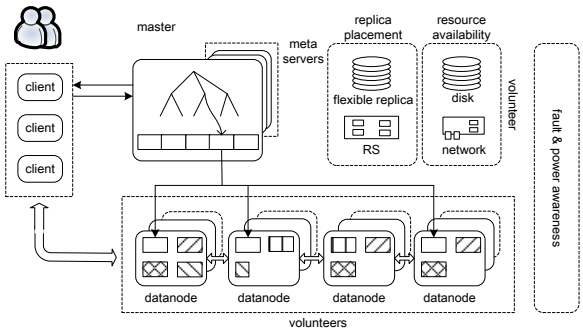



**Figure 1: Overall architecture**

RPC request will subtract part of budget according to its sending package size. When the budget is exhausted, next RPC request will be blocked until new budget is assigned in next second. Sometimes, we need to fine-tune the parameters to ensure no network peak is too high to hurt host's local applications.

Budget-based scheduling is also used to control disk IO bandwidth. But for multiple block devices, each 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 one standalone daemon, *self-manager*, which checks whether any over-consumption of resources occurred, and will execute suicide to clean malignant affect.

The *self-manager* is designed as a stand-alone daemon, so it can watch all contributory daemons and objectively audit their resource consumption. Different from hypervisor of virtual machine [2], it is more lightweight and quickly be launched or destroyed. Also different from kernel-level container (i.e. cgroup) [23], *self-manager* only control CPU and memory. Another key reason why not use virtual machine or kernel-level container is that almost all volunteer nodes are running online services, which does not allow 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, or cold data, encoded via Reed-Solomon (RS) algorithm [19][14]. Like [8], a big file will be separated as $256MB$-size data blocks, stored in distributed *datanode*. Each data block may have replication, or may be encoded together with several parity blocks. The classification of hot data and cold data results in that we can simply know which data are inactive and can save storage capacity by applying high compression algorithm.

The placement aims at reaching the balance of low cost and best availability. For three-replica hot data, one block replica is placed in high-quality storage media, other two mirrors are in cheaper media (one of them is trying to place on neighboring datacenter). For RS-encoded cold data, data blocks are trying to place on cheaper but better-performance medium, while parity blocks are placed in high-available medium. Therefore, accessing cold data will achieve better performance in most cases without data block crashes.

The placement of cold data helps data recovering to save net bandwidth. If data block crash happens, the recovery of cold data has to access all the data blocks and parts of parity

blocks. This would cause heavy IO consumption. To achieve the best recovering performance, an optional solution is to split cold data block into slices and dispersed them across several nodes. Fatman adopts RS(10, 4) to encode the data slices (RS($n, k$) is defined to represent one data is split to $n$ data blocks with $k$ parity blocks). Each slice owns 10 segments for data and 4 segments for parities. For un-failure data slice access, client only needs to read 10 data segments. However, when needing to tolerate $t$-failure block ($t < 4$), each slice will at most have 4 failure segments, therefore client will read $10 - t$ data segments and $t$ parity segments back in stream to recover the slice on crashed blocks. To recover one crashed data segment, other data segments will be pre-fetched from *datanode*, and the pre-fetched data segment can be used again to assemble the whole block. For hot data, read on failure data block will auto-switch to another replica simply.

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

## 3.4 Fault Awareness

To the best of our knowledge, accurate detection of storage failure can help system to optimize the mean time to repair (MTTR) [13], since the speed of repair after a failure determines reliability for a specific storage system [28]. However, classical erasure codes relied on by Fatman, like RS codes, are suboptimal for distributed environments because RS(10, 4) recovery requires transferring 10 blocks and recreating the original 10 data blocks even if a single block is lost [20]. Hence, the recovery process suffers a tenfold overhead in repair bandwidth and disk I/O. To be more efficient, Locally Repairable Codes (LRCs) has been introduced in [21][11][24], which gains 50% disk I/O and network reduction at the cost of 14% more storage.

As reported by [25], 78% of the hardware replacements in modern datacenters were caused by hard disk failures. So Fatman applies disk failure prediction for improve MTTR, and our result shows a better performance than LRCs without any more storage overhead. In our previous work [29], our model can predict 84.8% failures at least 24 hours ahead, preserving enough time margin for data migration from will-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 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 [7] 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 mean time to repair (MTTR) evidently. On the other hand, we may have an opportunity to migrate and recover data when 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 analyse the effects of this failure prediction method in Section 4.3.

# 4. EVALUATION

The goals of our evaluation here are (1) to illustrate 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; (3) to demonstrate the reliability impact of fault-aware data management and scheduling on MTTR of data repair and real-life product workload.

## 4.1 Resource Availability

We measure the resource consumption on one of real online volunteer nodes, which is configured with 8-core 2.4 GHz Xeon processors, 32 GB of memory, 12 1TB disks, and a 1 Gbps full-duplex Ethernet connection. Fatman *datanode* service and *self-manager* service are deployed within the same Linux account in the volunteer node. In Figure 2, the workload fluctuation (line *host*) is simulated from real local host application, to measure the impact on contributory services (*datanode*) about their resource consumption (line *contributory*). The measurement can be easily conducted via system tools or commands (e.g. *top* or *ps*).

CPU is the flexible resource and can be easily scheduled without killing processes. Usually, *datanode* keeps core consumption under specific given threshold (say 60%) when host workload is not heavy. When host workload increases and challenges more CPU resource, *datanode* will gradually return the core. In Figure 2(a), we simulate the workload changing as the fluctuation of dot-dash line. It can be seen from figure *datanode* withdrawing the resource consumption to ensure host's resource provision. When host workload continues to increase to upper limit, the *datanode* would be killed totally, which can be seen from the black line at 1000-second point, releasing all resources.

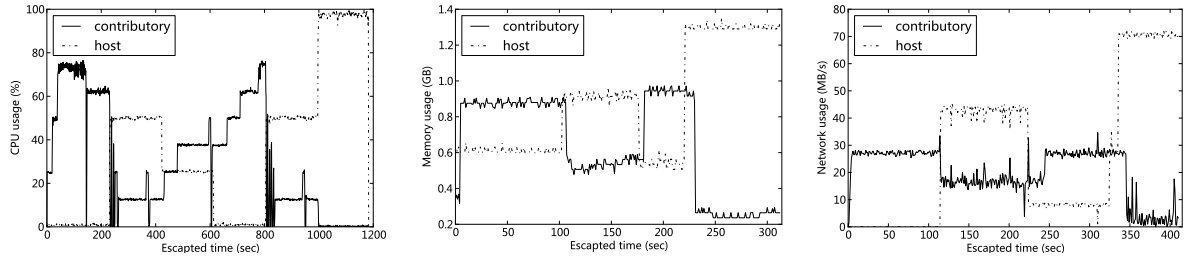
Differently, the memory is recycled via killing contributory services. We use *kill*, not *shut down*, because the execution is quicker and simpler to relase resources. The *self-manager* can be optionally restarted after suicide, by being configured in Linux *cron* service

The budget-based network scheduling can achieve the relatively stable utilization of contributory services. In our experiment, the network bandwidth is limited to 30 MB/s, we can see from Figure 2(c) that *datanode* will keep this level unless resource challenge is arrived at threshold from host workload. Similarly, *self-manager* detects the resource shortage and informs *datanode* to narrow down the bandwidth budget whenever necessary.

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

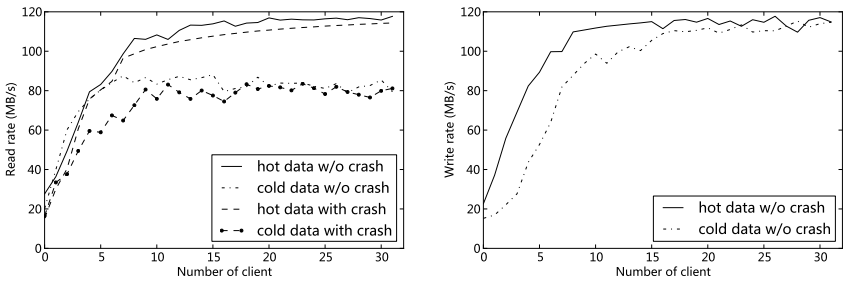
## 4.2 Performance

We measured performance on a Fatman cluster consisting of one master, two master replicas, 24 datanode servers, and 32 clients. Hardware configuration of each datanode servers is the same as real volunteer node. We prepare both hot data 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 ease of testing. Generally clusters have several thousands of datanode servers and thousands of clients.




(a) Elastic CPU usage (b) Elastic memory usage (c) Elastic network usage

**Figure 2: Elastic resource usage for dynamically changed workload**




(a) Reads (b) Writes

**Figure 3: Read/Write throughput**

### 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 real case. Figure 3(a) illustrates the aggregated read throughput with increase of client number. As for hot data with three replicas, read operation can easily reach $120MB/s$ peak limit of 1Gbps switch, and averagely $14MB/s$ for each client. But for RS-encoded cold data, the peak throughput fluctuates at about $85MB/s$ ($14MB/s$ for each client). Because the RS-decoding is streamingly processed from data segments across tens of *datanode* which swaps out some CPU time from networking, the curve in 3(a) cannot approach the bandwidth of hot data when increases the client's workload.

### 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 Figure 3(b). Averagely, write for both of hot data and cold data are reaching the peek $14MB/s$ per client. But the throughput for cold data can not 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 current implementation has to calculate the parity segment per ten data segments, there exists an idle scope of networking flow reducing the throughput. But the effect of idle scope can disappear gradually after $N > 16$ (see Figure 3(b)).

### 4.2.3 Reads on Failure Replication

Figure 3(a) also shows the performance of reads on those files which have some replicas are crashed. The experiment simulates the replica crash via random deleting replication without crashing whole file. For hot data, we random delete one or two replica(s), and for RS-encoded cold data, one or two block(s) are selected to be deleted. The missing reads can measure the overhead of data recovering when client tries to read back correct data from the system.

As shown in 3(a), the missing reads has minute influence on client throughput for hot data, which is decreased 0.4%. But for cold data it is 29.7% averagely. For hot data, client usually selects a closer and lower-workload replica to execute data read. If the replica is crashed or missing, the performance cost of client is only to switch to next replica. So, the the crash-replica read throughput of hot data is almost equal to no-crash reads. But for cold data, crashed replica would cause multiple additional read operations of parity segments to recovering the data segments. Plus the computation cost, there will decrease the throughput.

### 4.2.4 Data Recovering

Table 1 lists the recover performance of 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 the CPU and Memory. This may result in some performance loss during recovering. Since one crash/missing RS block will fetch tenfold data set, it needs to open ten connection averagely and read from geographically block file. Under the $40MB/s$ bandwidth, it needs 82.8 seconds to recover one block file.

**Table 1: Recover performance of single block**

| Type | Three-replica | RS-encoded |
|---|---|---|
| Block size ($MB$) | 256 | 256 |
| Trans. size ($MB$) | 256 | 256 * 10 |
| Opened connections | 1 | 10 |
| Network limit ($MB/s$) | 40 | 40 |
| Consumed time ($sec$) | 7.25 | 82.8 |

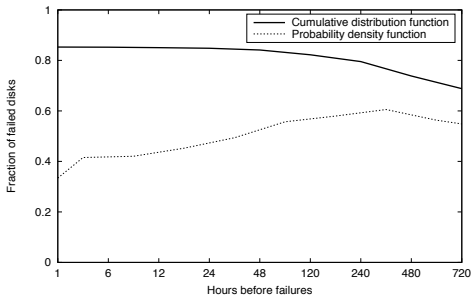



**Figure 4: The distribution of failed disks under different time margin**

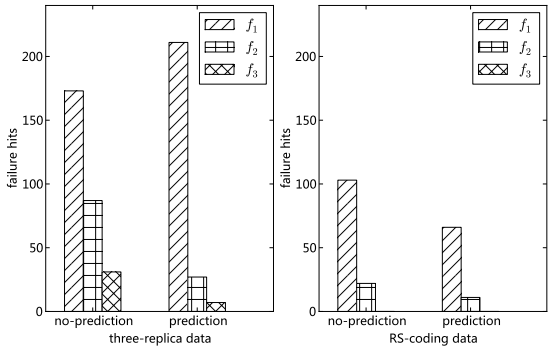



**Figure 5: The failure hits of read statics**

## 4.3 Fault Awareness

As discussed in Section 3.4, Fatman implements a fault-aware recovering mechanism by predicting incoming disk failures, which can reduce 76.3% of MTTR for RS recovery. For those will-fail drives that are predicted, 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, because the system just needs to migrate the data away from the will-fail drives, consuming only 1/10 repair time of RS recovery time. After the actual statistics, $1,384$ of the predictions were made 24 hours ahead of the 1632 failures, accounting for 84.8% of all the failed disks (shown by the cumulative distribution function in Figure 4). Therefore, the MTTR of RS should be reduced by $(1-1/10)*84.8\% = 76.3\%$ at least if the latent failures can be predicted 1 day ahead, saving the computing resources for RS decode in the meantime.

## 4.4 Reliability Improvement

We specifies some status to represent the file block changing. For three-replication hot data, $f_1$ means client hits one failure replication (including missing replica), $f_2$ means two hits, and $f_3$ means file has been crashed since at least one block in file has completely missed. The Figure 5 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. Compared no-prediction and prediction of hot data, we can found the failure prediction can help cut down 77% of crash replication ($f_3$) and 68% of two-replica failure ($f_2$). In Figure 5, the one-hits number of prediction is higher than no-prediction one, because two-failure-replication block will be recovered first which will increase the number of one-failure number.

For RS-encoded cold data, $f_2$ is specified to the status in which failure replication number is between two and $k$. The $k$ 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, so the $k$ is four. Therefore, total crashed replica on $f_1$ is less than those 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 produces are built based on computational resources like CPU and memory, since this category of runtime resources are relatively easier in control[15]. It introduces more challenges of volunteer storage to solve efficient utilization of heterogeneous host resources with varying capabilities and instability of high optional participation rate, which are two of basic requirements in volunteer computing[6].

Volunteer storage would cause SLO violation without efficient isolation during utilizing volunteer resources. VM-based isolation can achieve strong limitation but also introduce heavy overhead, while kernel-level container has not been ready for networking and disk IO[2][23]. Some still use the traditional audit-control methods in application level, while it is coarse-grained and insensitive[5][15]. However, the application-level control can be pervasive for not any requirements resident 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 proved, hard drive failure prediction can help maintain the reliability on storage system effectively. Merely relying on the S.M.A.R.T. standard (Self Monitoring and Reporting Technology) that most hard drive vendors will follow can predict at least 50 percent of future failures [7]. In order to further enhance the prediction accuracy, several machine learning techniques have been proposed using the SMART-based feature sets [9][12][17]. In our previous work [29], we also compared the performance of Backpropagation (BP) neural network and SVM.

Storage system has long tried to build more reliable system via disk mirroring [3] and RAID [4]. Erasure codes [18][19][14] 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 never reported in recent articles, especially applying prediction to pre-schedule decoding. Previous work focuses on self-monitoring detection, but not considering hardware-aware tolerance design.

# 6. CONCLUSION

Volunteer resource contribution is one of obvious ways to achieve high scalability and low investment for building large-scale archival system. However, traditional volunteer storage is hard to be popularized, which lies in the concerned influence on host applications against the reliable service

provision. The paper presents Fatman, a novel storage system design on volunteer contribution resources to build a reliable enterprise-scale archiving storage system. Fatman is implemented with strong resource limitation to maximally isolate influence against host applications. By fault-aware scheduler predicting potential hardware failure and perceiving QoS, Fatman can in-advance execute data scheduling and quality-aware data allocation, achieving high availability and reliability. The experiment result illustrates failure replication ratio has been reduced 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 100PB 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 data sets with periodic updates like webpage processing and user business logging.

# 7. REFERENCES


[1] File system archiving focus on EMC. `http://www.emcemea.com/materials/FileSystemAssess/ESG_FSA_Whitepaper.pdf`.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP'03*, pages 164–177, 2003.

[3] D. Bitton and J. Gray. Disk shadowing. In *VLDB'88*, pages 331–338, 1988.

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

[5] J. Cipar, M. D. Corner, and E. D. Berger. TFS: A transparent file system for contributory storage. In *FAST'07*, pages 215–229, 2007.

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

[7] J. f. Paris, J. f. Paris, T. J. E. Schwarz, T. J. E. Schwarz, D. D. E. Long, and D. D. E. Long. Evaluating the reliability of storage systems. Technical report, 2006.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *SOSP'03*, pages 29–43, 2003.

[9] G. Hamerly and C. Elkan. Bayesian approaches to failure prediction for disk drives. In *ICML*, pages 202–209, 2001.

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

[11] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in windows azure storage. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, USENIX ATC'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[12] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan. Improved disk-drive failure warnings. *IEEE Transactions on Reliability*, pages 350–357, 2002.

[13] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky. Are disks the dominant contributor for storage failures? a comprehensive study of storage subsystem failure characteristics. In *FAST'08*, pages 111–125, 2008.

[14] O. Khan, A. Burns, J. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: Minimizing i/o for recovery and degraded reads. In *FAST'12*, 2012.

[15] S. M. Larson, C. D. Snow, M. Shirts, V. S. P, and V. S. Pande. Folding@home and Genome@home: Using distributed computing to tackle previously intractable problems in computational biology.

[16] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for unix. *ACM Trans. Comput. Syst.*, pages 181–197, 1984.

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

[18] J. S. Plank. A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. *Softw., Pract. Exper.*, pages 995–1012, 1997.

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

[20] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. In *IPTPS'05*, pages 226–239, 2005.

[21] M. Sathiamoorthy, M. Asteris, D. S. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing elephants: Novel erasure codes for big data. *PVLDB*, pages 325–336, 2013.

[22] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? In *FAST'07*, pages 1–16, 2007.

[23] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. C. Bavier, and L. L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *EuroSys'07*, pages 275–287, 2007.

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

[25] K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. In *SoCC'10*, pages 193–204, 2010.

[26] M. Vrable, S. Savage, and G. M.Voelker. Bluesky: a cloud-backed file system for the enterprise. In *FAST'12*, 2012.

[27] M. Vrable, S. Savage, and G. M. Voelker. Cumulus: Filesystem backup to the cloud. In *FAST'09*, pages 225–238, 2009.

[28] Q. Xin, T. J. E. Schwarz, and E. L. Miller. Disk infant mortality in large storage systems. In *MASCOTS'05*, pages 125–134, 2005.

[29] B. Zhu, G. Wang, X. Liu, D. Hu, S. Lin, and J. Ma. Proactive drive failure prediction for large scale storage systems. In *MSST'13*, pages 1–5, 2013.