RAFI: Risk-Aware Failure Identification to Improve the RAS in Erasure-coded Data Centers

Juntao Fang, *Wuhan National Laboratory for Optoelectronics, Huazhong University of Sci. and Tech.*; Shenggang Wan, *School of Computer Science and Technology, Huazhong University of Sci. and Tech.*; Xubin He, *Department of Computer and Information Sciences, Temple University*

https://www.usenix.org/conference/atc18/presentation/fang

This paper is included in the Proceedings of the
2018 USENIX Annual Technical Conference (USENIX ATC '18).
July 11–13, 2018 • Boston, MA, USA
ISBN 978-1-939133-02-1

Open access to the Proceedings of the
2018 USENIX Annual Technical Conference
is sponsored by USENIX.

RAFI: Risk-Aware Failure Identification to Improve the RAS in Erasure-coded Data Centers

Juntao Fang[‡], Shenggang Wan[†], and Xubin He[§]
[†]*School of Computer Science and Technology, Huazhong University of Sci. and Tech., China*
[‡]*Wuhan National Laboratory for Optoelectronics, Huazhong University of Sci. and Tech., China*
[§]*Department of Computer and Information Sciences, Temple University, USA*
{yydfjt, sgwan}@hust.edu.cn, xubin.he@temple.edu

Abstract

Data reliability and availability, and serviceability (RAS) of erasure-coded data centers are highly affected by data repair induced by node failures. Compared to the recovery phase of the data repair, which is widely studied and well optimized, the failure identification phase of the data repair is less investigated. Moreover, in a traditional failure identification scheme, all chunks share the same identification time threshold, thus losing opportunities to further improve the RAS.

To solve this problem, we propose *RAFI*, a novel risk-aware failure identification scheme. In RAFI, chunk failures in stripes experiencing different numbers of failed chunks are identified using different time thresholds. For those chunks in a high risk stripe (a stripe with many failed chunks), a shorter identification time is adopted, thus improving the overall data reliability and availability. For those chunks in a low risk stripe (one with only a few failed chunks), a longer identification time is adopted, thus reducing the repair network traffic. Therefore, the RAS can be improved simultaneously.

We use both simulations and prototyping implementation to evaluate RAFI. Results collected from extensive simulations demonstrate the effectiveness and efficiency of RAFI on improving the RAS. We implement a prototype on HDFS to verify the correctness and evaluate the computational cost of RAFI.

1 Introduction

In large-scale erasure-coded data centers, node failures are the norm rather than the exception [1]. Those frequent node failures can result in numerous chunk failures (a chunk is the basic unit to organize data). The *RAS* (*R*eliability, *A*vailability, and *S*erviceability) of data centers are highly affected by repairing those failed chunks, which is known as *data repair*. Many solutions [2–19] are proposed to improve the RAS, i.e., reduce data loss, unavailability, and repair network traffic (a typical repair cost), through optimizing the data repair. However, existing solutions typically focus on the *recovery phase*, which is from the time when a chunk failure is identified to the time when the failed chunk is recovered. In contrast, the *identification phase*, which is from the time when a chunk failure occurs to the time when the chunk failure is identified, has not been explored yet. Consequently, the potential to further improve the RAS is not fully explored.

Traditionally, the failure identification of a chunk depends on the failure identification of its host node. When a node fails, its failure is not identified until a certain time threshold. When the node failure is identified, the failures of all the chunks on that node are identified, and the states of those chunks transition to *lost*. In summary, all chunks share the same time threshold with nodes in a traditional failure identification (TFI) scheme.

Under the TFI scheme, it is hard to simultaneously improve the RAS through adjusting the time threshold. On one hand, higher data reliability and availability could be achieved by lowering the failure identification time threshold, because of the shortened data repair time. On the other hand, the data center might suffer from increasing repair network traffic, because more transient node failures might be identified. In contrast, by increasing the failure identification time threshold, the repair network traffic could be reduced but the data reliability and availability might be suffer.

In this paper, we posit that the RAS can be simultaneously improved through optimizing the identification phase. This is rooted in the following dedicated observation on stripes. Each stripe consists of data chunks and parity chunks generated from those data chunks. A stripe is the basic unit for ensuring data reliability and availability. According to the number of failed chunks in a stripe, failed stripes can be classified into two types. One is a stripe which has many failed chunks, e.g., by default two or more failed chunks in a stripe with three parity chunks. This type of failed stripes is referred to as a high risk stripe. The other is referred to as a low risk stripe, which has a few failed chunks, e.g., by default one failed chunk in a stripe with three parity chunks. The more failed chunks a stripe has, the lower the data reliability and availability of the stripe are. Hence, most of the data loss and unavailability occur in high risk stripes. On the other hand, low risk stripes are much more common than high risk stripes, and thus induce most of the repair network traffic.

There already exist solutions that improve the RAS in the failure recovery phase, are rooted in being aware of the risk of stripes, e.g., prioritizing the recovery of the chunks in the stripes with multiple lost chunks [3, 7], or canceling the recovery of the chunks in the stripes with a few lost chunks [14]. Inspired by these approaches, we propose a novel *R*isk-*A*ware *F*ailure *I*dentification scheme, named *RAFI*, to improve the RAS of erasure-coded data centers. More specifically, RAFI is aware of not only lost chunks, which are focused on the traditional risk-aware wisdom, but also *unidentified failed* chunks, whose failure has not been identified yet. The key principle of RAFI is that the more failed chunks a stripe has, the shorter failure identification time threshold those chunks take. As a result, the aforementioned conflict between the data reliability and availability, and the repair network traffic is resolved, and the RAS are improved simultaneously.

We make the following contributions in this paper.

(1) We propose a risk-aware failure identification scheme RAFI to simultaneously improve the RAS of erasure-coded data centers. By deploying RAFI, a chunk failure is identified through multiple independent identification thresholds. Therefore, for chunks in high risk stripes, their failure identification is expedited, thus improving the data reliability and availability. For chunks in low risk stripes, their failure identification is postponed, thus reducing the repair network traffic. As a result, the RAS are improved simultaneously.

(2) A simulator is developed to verify our RAFI. The simulation results demonstrate that RAFI is very effective and efficient. For example, cooperating with all types of the state-of-the-art optimizations on the failure recovery phase, RAFI can further improve the data reliability by a factor of 9.3, and reduce the data unavailability and repair network traffic by 43% and 36%, respectively, at the cost of degraded reads increased by 1.7%.

(3) A prototype of RAFI is implemented in HDFS to verify the correctness and computational cost of our RAFI. The experimental results demonstrate that, in the worst case scenario, the computational cost of RAFI is still negligible.

The rest of this paper is organized as follows: Section 2 presents a model to analyze the relevance among the data reliability, repair network traffic, and failure identification. In Section 3, we give the design of RAFI. The results of prototype experiments and simulations are illustrated in Section 4 and 5, respectively. Section 6 reviews related work on optimizing the failure recovery phase, and Section 7 concludes the paper.

2 Background and Motivation

In this section, we first define the terms used in this paper. Then, we review the background materials of erasure-coded data centers, and summarize the existing methods to improve the RAS. Finally, we illustrate our motivation to propose RAFI.

2.1 Terms

Some terms to facilitate our discussion are summarized as follows.

A failed node: a node whose heartbeats have been lost. When a node fails, its heartbeat is lost immediately and it becomes a failed node. In TFI, the failure of a node is not identified until its heartbeats have been lost for over a certain time threshold.

A failed chunk: a chunk whose host node fails. When a node fails, all chunks on that node become failed. A failed chunk can be further classified into an unidentified failed chunk and a lost chunk as described below.

An unidentified failed chunk: a failed chunk whose failure has not been identified yet. Between the chunk failure occurs and that failure is identified, the chunk is treated as unidentified failed.

A lost chunk: a failed chunk whose failure is identified. After the failure of a chunk is identified, the chunk is treated as lost.

S^i and S^{i+}: a stripe S^i is a stripe with i lost chunks, and a stripe S^{i+} is a stripe with i and more lost chunks.

2.2 Erasure-coded Data Centers

To tolerate node failures, data redundancy techniques are usually deployed in data centers. Traditional data redundancy techniques, e.g., replication, suffer from high spatial cost. Hence, erasure coding techniques (e.g., Reed-Solomon coding) which have a much lower spatial cost compared to replication techniques, are widely used in data centers [7, 12, 20, 21].

To apply the erasure coding in data centers, data is first divided into fixed size data chunks. Then, parity chunks are generated from those data chunks. To prevent data loss or unavailability from node failures, all those data and parity chunks together form a stripe and are distributed to different nodes.

Table 1: Methods to Improve the RAS

	Time Threshold ↓	Recovery Penalty Factor ↑	Network Bandwidth ↑	Queue Time ↓
Reliability/Availability	↑	↑	↑	↑
Repair Network Traffic	↑	↓	→	→

Node failures are monitored through frequent heartbeats, e.g., every 3 seconds [3]. However, a node failure is not immediately identified when the heartbeats are lost, because most node failures are transient and those failed nodes can come back in a short period, e.g., 10 minutes [20]. In order to reduce the repair network traffic, only when the heartbeats have been lost over a certain time threshold, e.g., 15 minutes [20] or 30 minutes [7], a node failure is identified (a misidentification occurs if the node comes back).

Traditionally, when a node failure is identified, all the chunk failures due to that node failure are treated as identified failures. Surviving data and parity chunks (on other nodes) of the lost chunks would be fetched to repair those lost chunks (data repair), thus ensuring the data availability and reliability.

2.3 Methods to Improve the RAS

It is cost-effective to improve the RAS by optimizing the data repair process. Many solutions are proposed following this way which are explained below and also summarized in Table 1.

(1) Decreasing the time threshold reduces the repair time, and thus improves the reliability; however, it increases the repair network traffic;

(2) In erasure-coded data centers, multiple available chunks are transmitted over the network to recover lost chunks in the stripe. Recovery penalty factor is a factor which is between the amount data transmitted for recovering a stripe S^i and the size of a chunk. Decreasing the recovery penalty factor [2, 4, 5, 7–13, 16, 17, 22, 23] reduces the repair time, and thus improves the reliability; in the meanwhile, it reduces the repair network traffic;

(3) Increasing the network bandwidth [6, 24–26] of each storage node reduces the repair time, and thus improves the reliability; in the meanwhile, the repair network traffic stays almost the same.

(4) The queue time (waiting for recovery) of failed stripes is affected by recovery schemes. Giving high priority to S^i $(i > 1)$ [7, 27], the queue time of S^i $(i > 1)$ is decreased, and thus the reliability is improved; in the meanwhile, this method has little effect on the repair network traffic.

According to the above analysis and simulation results demonstrated in Figure 9a in Section 5.3, the RAS cannot be improved simultaneously by adjusting the failure identification time threshold. Therefore, a novel risk-aware failure identification scheme RAFI is proposed to explore the huge potential of simultaneously improving the RAS within the failure identification phase.

2.4 Motivation

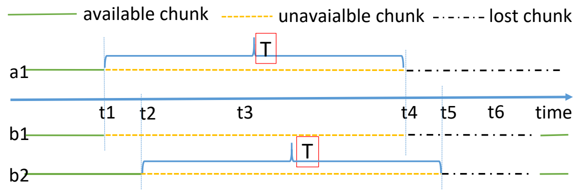
When some nodes fail, many stripes are affected, i.e., have failed chunks. Due to the randomized chunk layout, only a small fraction of those affected stripes have many failed chunks, and the remaining affected stripes only have a few failed chunks. Hence, most repair network traffic is induced by repairing the latter type of stripes.

On the other hand, the failure identification time of an arbitrary affected stripe having i failed chunks is equal to the failure identification time of its ith failed chunk, i.e., all the affected stripes share the same failure identification time. The stripes with many lost chunks usually entitle high recovery priority, i.e., a short queuing time. Hence, the repair time of those stripes are usually dominated by the failure identification time. In contrast, the stripes with a few identified failed chunks usually have a long queuing time. Hence, the repair time of those stripes are usually dominated by the recovery time.

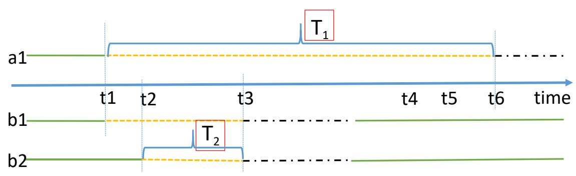
If the failure identification of those two types of stripes can be handled separately, the RAS of data centers can be improved simultaneously. More specifically, for the stripes having many failed chunks, we tune down the failure identification time threshold of those failed chunks, and thus improving the data availability and reliability at the cost of slightly increasing repair network traffic. For the stripes having a few failed chunks, we tune up the failure identification time threshold of those failed chunks, and thus reducing the repair network traffic without significantly reducing data reliability and availability. More importantly, the benefit induced by the above two operations would be dominant compared to the associated cost. Hence, the RAS of data centers can be improved simultaneously.

3 RAFI: Design and Analysis

In this section, we first present the design of RAFI; followed by a discussion on the benefit and cost of deploying RAFI.



(a) In TFI, a fixed threshold T is used to identify failures. The failure of chunk a1 is not identified until t4, while two failures of chunks b1 and b2 are not identified until t4 and t5, respectively.



(b) In RAFI, the failure of chunk a1 is identified through the threshold T_1 at t6, which is later than t4. On the other hand, the failures of chunks b1 and b2 are identified through the threshold T_2 at t3, which is ahead of t4 and t5.

Figure 1: Identification of chunk failures using TFI and RAFI. We use three sample chunks, where a1 is a random chunk of a stripe A while b1 and b2 are two random chunks of a stripe B. Assume chunk a1 fails at time t1 while chunks b1 and b2 fail at t1 and t2, respectively.

3.1 Design of RAFI

As we discussed above, the key problem of the traditional failure identification (TFI) scheme is that all chunks share the same failure identification time threshold. To simultaneously improve the RAS, we propose *RAFI* to identify chunk failures according to the risk level of their host stripes and apply different time thresholds accordingly. More specifically, dedicated chunk failure identification time thresholds are set for stripes in different risk levels, which are determined by the *total failed chunks* in the stripes. For chunks in low risk stripes, long failure identification time thresholds are adopted, thus reducing the repair cost. For chunks in high risk stripes, short failure identification time thresholds are adopted, thus improving the data reliability and availability. As a result, the RAS are simultaneously improved.

In summary, the key design principle of RAFI is that the more failed chunks a stripe has, the shorter failure identification threshold those chunks take. For a failed chunk in a stripe with i failed chunks, there are at most i identification thresholds to identify the failure of this chunk, and the jth ($0 < j \leq i$) identification threshold is described as follows. If there are j failed chunks and the failure durations of these j failed chunks are all longer than a preset time threshold T_j, all these j chunk failures are identified and these chunks are denoted as lost immediately. The state of an unidentified failed chunk in these j chunks transitions to lost, and a lost chunk in these j chunks remains lost. The states of the remaining $(i-j)$ chunks do not transition.

In RAFI, a chunk failure is identified by independent identification thresholds, which is quite different from the traditional single identification threshold described in Section 1. For example, in a (6,3)-coded data center, stripe A has one failed chunk and is a low risk stripe, stripe B has two failed chunks and is a high risk stripe. A time threshold T_1 which is larger than the original time threshold T is set to identify failures of chunks in the low risk stripe; while a time threshold T_2, which is shorter than the T is set to identify failures of chunks in the high risk stripe. As shown in Figure 1, using RAFI, the failure identification of chunk a1 in the stripe A is postponed; in the meanwhile, the failure identification of chunks b1 and b2 in the stripe B is expedited.

RAFI is flexible. First, all the time thresholds can be set independently to get proper trade-offs between the data reliability and availability, and the repair network traffic for a certain type of stripes. Second, the identification thresholds can be merged to get proper trade-offs between the RAS and the computation cost of RAFI. When the time thresholds in all identification thresholds are set to the same T, RAFI becomes TFI.

3.2 Benefit and Cost

Improving the RAS: Using RAFI, we can independently set different time thresholds to identify failures. First, short thresholds are used to expedite the identification of failed chunks in high risk stripes, thus improving the data reliability and availability. At the same time, long thresholds are used to postpone the failure identification of chunks in low risk stripes, thus reducing the repair network traffic and improve the serviceability. Because the identification time is dominant in the repair time of chunks in high risk stripes, the expedition is effective in improving the data reliability and availability thus compensates the negative impacts induced by the postponement. Because most repair network traffic is induced by recovering chunks in low risk stripes, the repair network traffic is significantly reduced, even under the consideration of the extra repair network traffic induced by the expedition, thus improving the serviceability.

Compatibility: Because RAFI focuses on the failure identification phase, it can work together with existing optimizations which focus on the failure recovery phase.

Increasing Degraded Reads: Degraded read is an operation to read unavailable but recoverable chunks in a stripe. Because we postpone the failure identification of chunks in low risk stripes, more failed chunks might be generated, thus increasing degraded reads. However, the simulation results in Section 5 show that degraded reads increase by less than 1.7% due to RAFI. Because degraded reads are much fewer than normal reads, the overhead is very small.

4 Prototyping Evaluation

In this section, we first present the evaluation methodology; then we illustrate implementation details of our prototype RAFI-HDFS; finally we demonstrate results of prototyping experiments.

4.1 Evaluation Methodology

To verify the effectiveness of RAFI, we propose a hybrid methodology to comprehensively evaluate RAFI via both simulation and prototype implementation. The reason is explained below.

It is difficult to evaluate a technique targeting at the RAS of data centers because the data loss and unavailability events are very rare and not evenly distributed. The accuracy problem induced by that uneven distribution can be mitigated by high accurate simulation, which is run thousands to millions of iterations, although the simulator itself might be not that accurate. However, pure simulation cannot verify the correctness of design details and might cover fatal defects of the technique.

In our hybrid evaluation, the design details and computational cost of RAFI are verified through prototyping running on a real distributed storage system; the effectiveness and efficiency of RAFI on the RAS are evaluated through high accurate Monte-Carlo simulation.

In this section, we evaluate the identification time and computational cost of RAFI in real world environments. The simulator and simulation results are discussed in Section 5.

4.2 RAFI-HDFS

To evaluate the effectiveness of RAFI, we implement a prototype named RAFI-HDFS on HDFS [27], a representative distributed file system widely deployed in the data centers. Because erasure coding is supported by HDFS in version 3.0.0, our implementation is based on HDFS 3.0.0-alpha2. The implementation of RAFI-HDFS follows the design in Section 3. We only add about 200 lines of codes to HDFS.

Figure 2 demonstrates the overall architecture of RAFI-HDFS consisting of two modules: one is a classification module and the other is an identification module.

The classification module is responsible for converting the node failures into appropriate input for the identification thresholds. More specifically, the classification module receives a node list that contains all failed nodes and their failure durations from the node monitor module. Only those nodes whose failure durations are larger

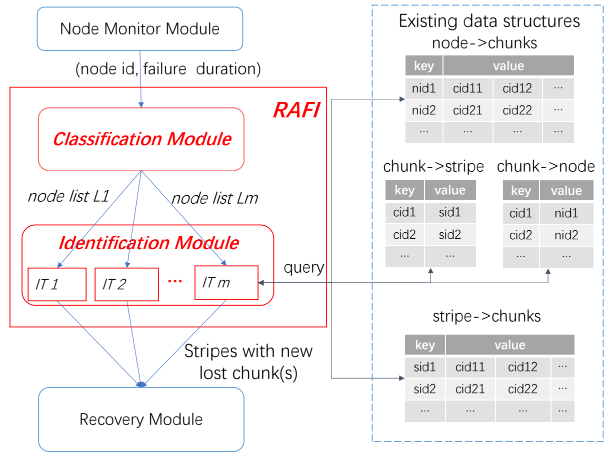


Figure 2: Architecture of RAFI-HDFS. The right side is existing data structures which are used in RAFI. The node monitor module reports failed nodes and their failure durations. The classification module inserts nodes to different identification thresholds in the identification module according to their failure durations. The identification thresholds (IT) in the identification module are used to identify chunk failures.

than T_i ($1 \leq i \leq m$) are inserted into the node hash list L_i for the *identification threshold (IT) i*, thus reducing the computation cost of that identification threshold. It is worth noting that the classification module replaces failed chunk lists with failed node lists. In such a manner, the memory usage of maintaining the numerous failed chunks is saved.

The identification module is a universal set of all the identification thresholds in RAFI. When *IT i* receives its node list L_i, it begins to calculate the *count* of failed chunks in stripes. First, the identification threshold calculates the *count* of unidentified failed chunks in stripes through querying the node-chunk mapping table and the chunk-stripe mapping table, which typically reside in the main memory of the manager nodes of the data centers. Second, through querying the stripe-chunk mapping table and chunk-node mapping table, the *count* of lost chunks is obtained. If the count of failed chunks (unidentified failed chunks and lost chunks) is larger than or equal to i, those failed chunks which belong to nodes in L_i, transition to lost.

After working through all identification thresholds, if new chunk failures are identified, the recovery module is called to recover stripes containing those lost chunks. Particularly, for nodes which enter *IT* 1, the failures of these nodes are identified and these nodes are removed from the system at the end of the *IT* 1.

Complexity. RAFI-HDFS only checks chunks on failed nodes which newly enter L_i to reduce the computation cost. Assume there are j nodes in L_i ($2 \leq i \leq m$) and there are an average of d chunks to be checked on the node. Each stripe has $k+m$ chunks. Because we use a hash list to track the failed nodes, the total comparison time is about $(k+m) \times d$. The time complexity of identifying chunk failures is $\mathcal{O}(d)$.

4.3 Results of Prototyping Experiments

Experimental Setups. The experimental system consists of ninety-seven servers running on the Alibaba Cloud [28]. One server served as a NameNode contains an Intel Xeon E5-2682v4 @ 2.5 GHZ CPU (4 vCPU), 16 GB DDR4 memory, 1.5 Gbps network and 40 GB SSD. The remaining 96 servers are used as DataNodes, each of which has an Intel Xeon E5-2680v3 @ 2.5 GHZ CPU (1 vCPU), 1 GB DDR4 memory, 1 Gbps network and 40 GB SSD. The operating system running on all these servers is Ubuntu 14.04. Each DataNode sends heartbeats to the NameNode every 3 seconds and the NameNode checks the states of all DataNodes every 5 minutes. As default in HDFS, the time threshold $T = 10.5$ minutes and the erasure coding scheme RS(6,3) is used.

Identification Time of Chunks: The identification time of a chunk is the period from the time when a chunk becomes failed to the time when the chunk is identified as a lost chunk. In order to evaluate the real identification time, we collect the identification times by randomly killing two DataNodes. In order to evaluate the real identification time of chunks, we collect the identification times by randomly killing DataNodes in 0, 5, 10, and 20 minutes. Each experiment is conducted 20 times. In RAFI, T_2 is set to 1 minute and T_1 is set to 60 minutes. The results are consistent with our analysis in Section 3.2. The results demonstrate that TI_2 is expedited and TI_1 is postponed. When we simultaneously kill two storage nodes, TI_1 and TI_2 under TFI are 13.1 minutes; however, TI_2 under RAFI is 3.6 minutes, while TI_1 under RAFI is 62.6 minutes. Moreover, TI_1 and TI_2 are not relevant to the time between the failure arrivals.

Burden on the NameNode: Because the computations run on the NameNode, we record the time spent to identify chunk failures when nodes fail to further estimate the impact on the NameNode. The chunk size is shrunk to $1KB$ in our cluster to generate enough number of chunks. In the experiments, each DataNode stores about 68,000 chunks. In the experiments, there is no I/O workloads because the time spent to identify chunk failures under no I/O workloads is sufficient to indicate the overhead caused by RAFI on the NameNode. For each result, we concurrently kill DataNodes. Each experiment is conducted 10 times and we calculate the average results.

We evaluate the time spent to identify chunk failures from two aspects: the number of DataNodes in the cluster and the number of concurrent node failures.

First, as shown in Figure 3, the time spent to identify all 68,000 chunk failures on one failed DataNode increases from 74 to 137 milliseconds when the number of DataNodes increases from 12 to 96. Compared to time thresholds and check intervals (by default 10.5 and 5 minutes, respectively), the time spent to identify chunk failures can be negligible in the identification time.

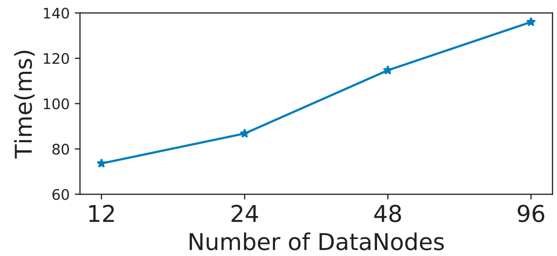


Figure 3: Time spent to identify chunk failures when when a DataNode fails. The number of DataNodes in the cluster changes from 12 to 96. E.g., the NameNode takes 87 ms to identify 68,000 chunk failures in a cluster of 24 DataNodes.

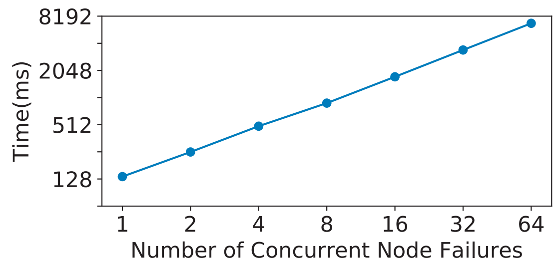


Figure 4: Time spent to identify chunk failures when DataNodes fail. The cluster consists of 96 DataNodes. E.g., the NameNode spends 889 ms to identify 544,000 chunk failures when eight DataNodes fail concurrently .

Second, as illustrated in Figure 4, the time spent to identify chunk failures increases linearly as concurrent node failures increase. The experiment results are consistent with the analysis in Section 4.2. It is worth noting that there are no failed nodes in most check time. Thus, our method has minimal impact on the NameNode.

Moreover, in our evaluation, only one single thread is used to check all chunks on failed nodes. In fact, we can use multi-threading technologies to check all chunks on failed nodes, e.g., each thread is responsible for checking all chunks on one failed node. Therefore, the time spent to identify all chunks on failed nodes can be dramatically reduced when multiple nodes fail concurrently.

5 Simulations and Results Analysis

In this section, we discuss our simulator and simulation results to evaluate the effectiveness and efficiency of RAFI on the RAS.

5.1 DR-SIM

We developed a simulator called DR-SIM which is written in the R language because it easily runs in parallel. The source code is approximately 1400 lines [29].

Event-driven simulators are widely used to study the RAS of data centers [14, 20, 30]. However, those simulators cannot be used in our simulations due to the following two reasons. First, some simulators are not open source, e.g., the Google's Cell Simulator [20]. Second, the RAS cannot be all simulated by some simulators. For example, limited by performance, the data reliability cannot be studied by the ds-sim [14]. As a result, we develop our own simulator, named DR-SIM, to study the effect of the data repair on the RAS in data centers.

We summarize important features of DR-SIM as follows. (1) The trade-off between the performance and accuracy of DR-SIM is carefully tuned. A simulation iteration (typically represents five years) can be finished in tens of seconds. Therefore, we run hundreds of thousands iterations for each simulation configuration, to accurately measure the RAS. (2) Many state-of-the-art optimizations on the data repair are integrated into DR-SIM, and important parameters of the data repair are considered as variants in DR-SIM. Through modifying the configuration of DR-SIM, we study the effectiveness and efficiency of RAFI upon various combinations of the state-of-the-art optimizations under various typical environments of the data centers.

Figure 5 shows the architecture of DR-SIM which includes four modules: a configuration manager, a failure generator, a repair calculator, and an event collector.

The configuration manager loads parameters used in the simulations. The parameters are explained as follows. (1) *System parameters*: The target erasure-coded data center consists of N independent storage nodes. Each node has d chunks. The chunk size is s. (2) *Coding parameters*: Data are coded with (k, m) erasure codes, i.e., k data chunks and m parity chunks are in a stripe. The $k+m$ chunks in the same stripe are distributed to $k+m$ distinct nodes. A random placement policy is used because it is usually adopted in practice. The recovery penalty factor of S^i ($1 \leq i \leq m$) is r_i which is between the amount data transmitted for recovery of S^i and s. The recovery network bandwidth is b on each node. (3) *Failure parameters*: Assume node failure arrivals are independent. Let $f(x)$ and $F(x)$ be the probability and cumulative distribution functions of the failure arrivals, respectively. Assume failure durations are independent. Let $g(x)$ and $G(x)$ be the probability and cumulative distribution functions of the failure durations, respectively. ρ is the ratio of permanent node failures to all node failures. τ denotes the additional proportion of correlated node failures. (4) *Identification parameters*: Storage nodes periodically send heartbeats to dedicated manager nodes, e.g. the NameNode [27, 31] or the metadata manager [32]. The manager nodes check states of all nodes at regular time intervals of T_h. The time thresholds for identifying chunk failures are T_i ($1 \leq i \leq m$). (5) *Simulation runtime parameters*: N_i denotes the number of iterations. T_d is the simulation duration for each iteration.

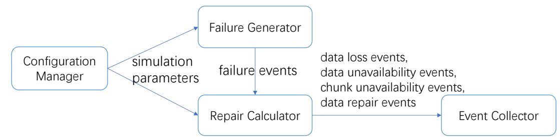


Figure 5: Architecture of DR-SIM

The failure generator is responsible for generating failure arrivals and failure durations of node failures at the beginning of a simulation iteration. The failure arrivals are generated according to the distribution function $f(x)$. Permanent failures and transient failures are generated by their durations. For the transient failures, their durations are generated according to the distribution function $g(x)$. For permanent failures, they are generated according to the parameter ρ. Technically, failure durations of the permanent failures are set to zero (only for handling but not calculating). In DR-SIM, additional correlated failures are explicitly generated by adding a random value between 0 to 120 seconds [20] to existing failure arrivals according to the parameter τ. It is worth noting that the comeback of transient failed nodes has been already considered in DR-SIM.

The repair calculator simulates the data repair process for lost chunks when failures occur. The repair calculator identifies the chunk failures according to the T_h and T_i ($1 \leq i \leq m$) and calculates the repair time for lost chunks based on the recovery network bandwidth, the recovery penalty factors and the recovery priority. The recovery processes of lost chunks are scheduled depending on the number of lost chunks in their stripes. For stripes have the same number of lost chunks, the repair calculator uses first come first scheduled rule to manage the recovery of those chunks. Moreover, lost chunks are recovered in parallel by utilizing all available nodes [33, 34].

The event collector is responsible for collecting data loss events, data unavailability events, chunk unavailability events, and data repair events. At the end of each iteration, DR-SIM calculates metrics according to the collected events. The mean time to data loss in the whole data center (referred as ***MTTDL***) is the metric to evaluate the data reliability. All the data loss events are recorded to calculate the *MTTDL*. The cumulative unavailable time of all stripes (referred as T_{us}) is the metric to evaluate the data availability. All the data unavailability events are recorded to calculate the T_{us}. The total repair network traffic (referred as ***RNT***) is the metric to evaluate the serviceability. All the data repair events are recorded to calculate the *RNT*. The cumulative unavailable time of all chunks (referred as T_{uc}) is the metric to evaluate the degraded reads. All the chunk unavailability events are recorded to calculate the T_{uc}. The former three metrics are widely used in evaluation of the RAS in the data centers [6, 7, 12, 14, 15, 20, 30, 35, 36]. The latter one is roughly in proportion to the number of degraded reads. It is worth noting that chunks and stripes are actually not simulated in DR-SIM under the consideration of computation complexity. In fact, the cumulative unavailability time of stripes and cumulative unavailability time of chunks are estimated from the generated node failures and data repair events.

5.2 Simulation Testbed

Comparisons between RAFI and TFI are made upon the testbed described as follows.

The following three state-of-the-art optimizations are always considered in the testbed. (1) The network adopts CLOS topologies [24–26]. (2) All lost chunks are parallel recovered via using available recovery network bandwidth on all nodes. (3) The chunks in stripes with more lost chunks have the higher priority to be recovered.

Three kinds of erasure codes are chosen in the simulations to understand the sensitivity to different erasure codes. RS codes are are a set of popular erasure codes which are widely used in real world distributed storage systems [12, 20, 21]. Zigzag codes [10] represent MDS (Maximum Distance Separable) codes with optimal recovery penalty factors. LRC codes [7] are representative non-MDS codes deployed in Windows Azure Storage.

The 1 Gbps network is chosen as the baseline in the testbed under the consideration of the cost-effectiveness in the erasure-coded data center, although we have found that RAFI is more efficient in reducing the *RNT* under the 40 Gbps network during studying the sensitivity of RAFI to the recovery network bandwidth.

Because chunks in low risk stripes are the optimization objects of both RAFI and *Lazy* [14], Lazy is considered in the testbed when we made dedicated comparisons between these two techniques in Section 5.3.4.

Default values of most parameters used in the simulations are listed in Table 2. The failure arrivals are assumed to be independent and exponentially distributed with the mean time to failure (MTTF = 7.1 days) [12,20]. The failure durations are assumed to be independent and Weibull distributed. We get sample values from [20] and model the failure durations with Weibull(113 seconds, 0.54), which is shown in Figure 6. The model fits well starting from 0.5 minutes.

In our simulations, to simplify the comparison complexity, the identification thresholds *identification threshold i* ($i > 1$) are merged to one by sharing the same threshold value. The features of the erasure codes, and two time threshold values (one for T_1, and the other for T_i ($i > 1$)) are represented by an abbreviation, e.g., RS(6,3)-15-2 denotes a data center employed RS(6,3) with T_1 = 15 minutes and $T_2 = T_3$ = 2 minutes. r_1, r_2 and r_3 of an RS(6,3)-coded stripe are 6, 7, and 8, respectively. All measured metrics including the *MTTDL*, T_{us}, *RNT* and T_{uc}, are normalized to that of the RS(6,3)-15-15 (it denotes a TFI configuration when the latter two values are the same). The *MTTDL*, T_{us}, and *RNT* are the metrics to evaluate the RAS.

Table 2: Symbols and Their Definitions

Symbol	Definition	Default Value
N	# of storage nodes in a data center	1000
d	# of chunks on a node	125,000
s	Chunk size	128 MB
T_h	Check interval of node states	5 minutes
b	Recovery network bandwidth on each node	0.1 Gbps
T_d	Duration of each iterations	5 years
N_i	# of iterations	500,000

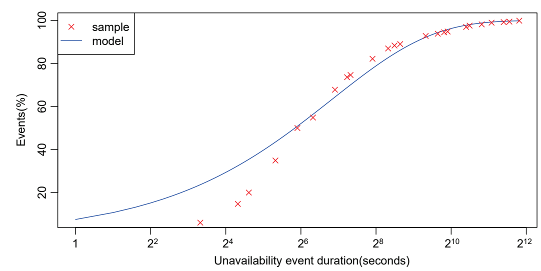


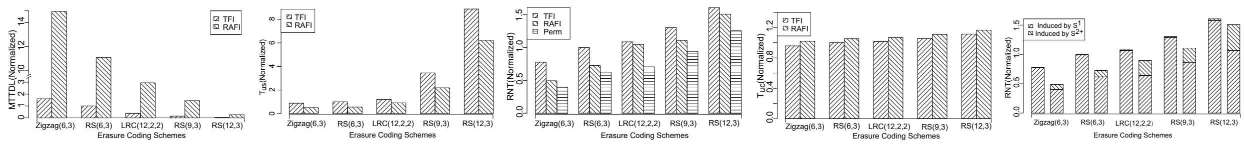
Figure 6: Unavailability Event Duration

5.3 Simulation Results

5.3.1 RAS as Functions of T_i

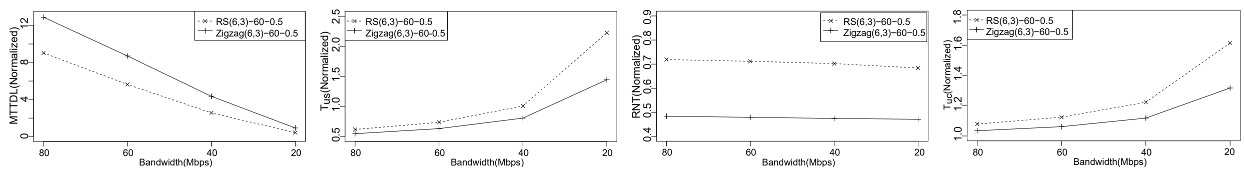
First of all, we run simulations to find the proper two threshold values for RAFI. Let $T_3 = T_2 = T_1$. Figure 9a illustrates that the data reliability and availability get worse while the repair network traffic is improved when T_1 increase. The *RNT* reduces slowly when T_1 is larger than 60 minutes. Thus, T_1 of RAFI is set to 60 minutes in the rest simulations.

Then, to study the impact of T_2, let $T_3 = T_2$. T_2 ranges from 0.5 to 8 minutes. The results in Figure 9b demonstrate that RAFI simultaneously improves the RAS in most configurations. More specifically, the *MTTDL* is improved by a factor up to 11. The T_{us} is reduced by up to 45%. The *RNT* is reduced by up to 27%. The *RNT* increases with the reduction of T_2 because reducing T_2 increases the number of S^{2+}, and results in unnecessary



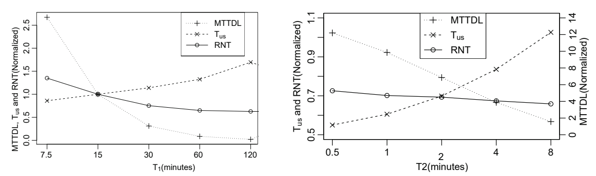
(a) Reliability (b) Availability (c) Serviceability (d) Degraded Reads (e) *RNT* induced by S^1 v.s. *RNT* induced by S^{2+}.

Figure 7: Impacts of different erasure coding schemes on the RAS. The results are normalized to RS(6,3)-15-15.



(a) Reliability (b) Availability (c) Serviceability (d) Degraded Reads

Figure 8: Impacts of constrained recovery network bandwidth on the RAS. RS(6,3) and Zigzag(6,3) are considered in the simulations. The results are normalized to RS(6,3)-15-15.



(a) The *MTTDL*, T_{us}, and *RNT* with different T_1. (b) The *MTTDL*, T_{us}, and *RNT* with different T_2. T_1 is set to 60 minutes.

Figure 9: Impacts of T_1 and T_2. The erasure coding scheme is RS(6,3), and the results are normalized to RS(6,3)-15-15.

repair network traffic to repair those S^{2+}. Only when T_2 is 8 minutes, which is close to the original T of 15 minutes, RAFI does not take effect on the data availability.

From the results, we find that the data reliability and availability are sensitive to the decrease of T_2 but the repair network traffic is not sensitive to the decrease of T_2. As a result, *both* T_2 *and* T_3 *are set to 0.5 minutes* in the rest simulations.

5.3.2 RAS as Functions of Erasure Coding Schemes

In this section, we examine the effectiveness and efficiency of RAFI under five typical kinds of erasure coding schemes, RS(6,3), RS(9,3), RS(12,3), Zigzag(6,3) [10], and LRC(12,2,2) [7]. These erasure coding schemes represent various recovery penalty factors. T_1, T_2 and T_3 are 60 minutes, 0.5 minutes and 0.5 minutes, respectively. All results are normalized to RS(6,3)-15-15 and presented in Figure 7. In general, RAFI can cooperate with all the five kinds of erasure coding schemes, and simultaneously further improve the RAS at the cost of the slightly increased degraded reads.

Improving Reliability: Figure 7a shows that RAFI improves the *MTTDL* of Zigzag(6,3), RS(6,3), LRC(12,2,2), RS(9,3), and RS(12,3) by a factor of 9.3, 11, 7.7, 9.8, and 7.7, respectively. When the recovery penalty factor increases, the improvements diminish a little. The reason is that the higher recovery penalty factor lengthens the recovery time, thus weakens the effect of the reduction of the identification time.

Improving Availability: Figure 7b illustrates that RAFI improves the data availability under various erasure coding schemes. The T_{us} of Zigzag(6,3), RS(6,3), LRC(12,2,2), RS(9,3), and RS(12,3) is reduced by 43%, 45%, 24%,37%, and 30%, respectively.

Improving Serviceability: Figure 7c shows that RAFI reduces the *RNT* under various erasure coding schemes. The *Perm* represents the *RNT* induced only by permanent node failures. Figure 7e shows the composition of the *RNT*. In TFI, over 99% of the *RNT* is induced by the repair of S^1. In RAFI, about 15%-30% of the *RNT* is induced by the repair of S^{2+}.

Degraded Reads: When RAFI postpones the recovery of S^1, the amount of unidentified failed chunks increases. Figure 7d shows that the degraded reads increase by 1.7% at most, which is very slight.

5.3.3 RAS as Functions of Recovery Network Bandwidth

Network bandwidth is very valuable in the data centers. In this section, simulations are performed to understand the effect of RAFI under a limited recovery network bandwidth b. Both RS(6,3) and Zigzag(6,3) codes are considered in the simulations. T_1, T_2 and T_3 are 60 minutes, 0.5 minutes and 0.5 minutes, respectively. The simulation results are normalized to RS(6,3)-15-15 and presented in Figure 8.

Figure 8 shows that the RAS are still improved even when b is 40 Mbps. However, at the same time, the T_{uc} increases by 22%, because a small b significantly extends the repair time of the lost chunks, thus leads to longer chunk unavailability time. When b reduces, the reduction of *RNT* increases a little.

Table 3: The RAS improvements under 40 Gbps network

Erasure Coding Schemes	RS(6,3)	Zigzag(6,3)
Improvement of *MTTDL*	3.4	3.7
Reduction of T_{us}	54%	56%
Reduction of *RNT*	79%	86%

40 Gbps network: Nowadays, some data centers are equipped with 40 Gbps network for each node [26, 37]. In such a scenario, the recovery network bandwidth b is 4 Gbps for each node. Table 3 shows that RAFI still improves the RAS when b is 4 Gbps. When b increases from 100 Mbps to 4 Gbps, the recovery time reduces. Because the ratio between the recovery time and the repair time decreases, the improvement of *MTTDL* decreases. However, when the repair rate increases, there will be more unnecessary repair network traffic. Therefore, RAFI is very effective in reducing the repair network traffic.

5.3.4 Comparisons with *Lazy*

To comprehensively compare RAFI with *Lazy*, the comparisons are made in the form of TFI + *Lazy* v.s. RAFI + *Lazy* v.s. RAFI. RS(6,3) and Zigzag(6,3) codes are considered in the simulations. *Lazy* [14] recovers lost chunks if their host stripes have at least two lost chunks. In TFI + *Lazy*, we use the parameters: $T_1 = T_2 = T_3 = 15$ minutes. In RAFI + *Lazy*, $T_1 = T_2 = 15$ minutes, $T_3 = 1$ minutes. In RAFI, $T_1 = 60$ minutes and $T_2 = T_3 = 15$ minutes. The comparison results are shown in Figure 10.

Cooperating with *Lazy*, compared to TFI, RAFI improves the *MTTDL* by a factor of 5.1, at the cost of increasing the *RNT* by 2.5%. Because *Lazy* even does not recover some permanent failed chunks, RAFI cannot further reduce the *RNT*.

Compared to TFI + *Lazy*, RAFI without *Lazy* increases the *MTTDL* by over two orders of magnitude at a higher *RNT* cost. An interesting thing is that, RAFI suffers a much lower increase of the *RNT* when cooperating with the Zigzag codes. The reason is that the recovery penalty factor of a Zigzag(6,3)-coded S^1 is only 63% of that of an RS(6,3)-coded S^1. In fact, as mentioned in Section 6,

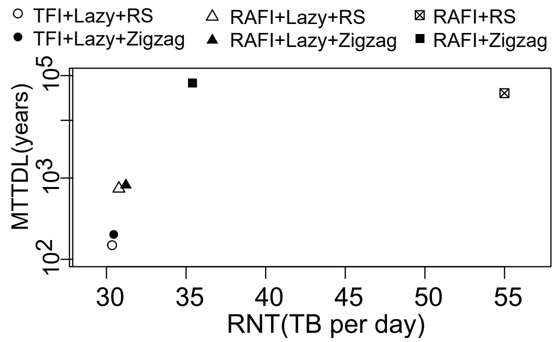


Figure 10: The *MTTDL* and *RNT* under TFI+*Lazy*, RAFI+*Lazy*, and RAFI. The erasure coding schemes are RS(6,3) and Zigzag(6,3). X axis is the repair network traffic, Y axis is the *MTTDL*.

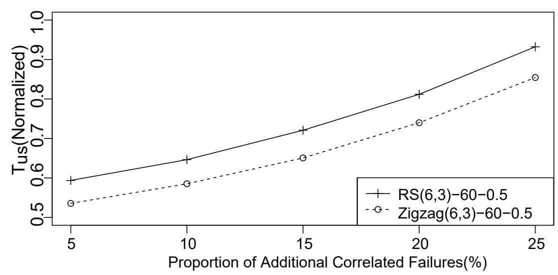


Figure 11: Impact of correlated failures on availability. The results are normalized to RS(6,3) with no additional correlated failures.

many codes [6, 7] are proposed to reduce the recovery penalty factor of stripes with one lost chunk.

5.3.5 Availability under Correlated Failures

Because transient failures may happen concurrently [20], we desire to see how data availability is affected by correlated failures. From Figure 11, we can see that, as the proportion of additional correlated failures increases, RAFI still reduces about 40% of the T_{us}, demonstrating that RAFI is very resilient to correlated failures.

6 Related Work

Existing solutions which are proposed to improve the RAS focus on optimizing the failure recovery phase, such as reducing recovery penalty factors [2, 4, 5, 7–13, 16, 17, 22, 23], improving recovery rates [6, 18, 19], and risk-aware recovery scheduling [3, 7, 14].

Reducing recovery penalty factors: Both the recovery time and repair network traffic are improved through reducing the recovery penalty factors of erasure codes. Two types of techniques are proposed. One is to design MDS and non-MDS erasure codes with low recovery penalty factors [2, 6–12, 15–17, 38]. The other is to design recovery algorithms to reduce recovery penalty factors of existing erasure codes [4, 5, 13].

Regenerating Codes [22, 23, 38, 39] are a family of MDS codes. The recovery penalty factors of the Regenerating Codes are much lower than that of the traditional RS (Reed-Solomon) codes [40]. However, the Regenerating Codes are not systematic codes, thus suffer from high read cost. To maintain low recovery penalty factors and read cost, systematic MDS codes, such as Zigzag and Butterfly codes [10, 17] are proposed. Zigzag codes [10] are proved to be with optimal recovery penalty factors in all systematic MDS codes. One significant drawback of Zigzag codes is that the implementation depends on non-binary algebra.

New trade-off points between storage overheads and recovery penalty factors are found through non-MDS codes, such as LRC [7, 11, 16]. Compared to MDS codes, non-MDS codes dramatically reduce the recovery penalty factors. However, the cost of non-MDS codes cannot be ignored, particular when the scale of the data center is very large, i.e., even 1% extra storage overhead usually means millions of dollars [41, 42].

Recovery algorithms, such as [4,5,13], are proposed to reduce recovery penalty factors of existing erasure codes. The biggest drawback of those recovery algorithms is that their efficiency on reducing recovery penalty factors are much lower than that of designing novel codes.

Improving the recovery rate: Another approach to shorten the recovery time is improving the recovery rate.

It is common to improve the recovery rate through deploying high-speed networks, i.e., increasing the recovery network bandwidth. For example, CLOS networks [24–26] are deployed in FDS [6] to provide non-oversubscribed full bisection bandwidth networks at the scale of a data center. As a result, the recovery is dramatically accelerated.

The recovery rate is also improved through increasing the recovery parallelism. Mitra et al. propose a parallel chunk recovery method PPR [18] to improve the recovery parallelism. Li et al. propose a pipelined chunk recovery method ECPipe [19] to further improve that recovery parallelism. However, both PPR and ECPipe take effect when there are only a few chunks be recovered.

Risk-aware recovery scheduling: Besides accelerating the recovery of all chunks, high data reliability and availability can also be achieved through scheduling the recovery of chunks according to the *number of lost chunks* in their host stripes, which indicates the data reliability and availability risk of those stripes.

The recovery of the chunks in high risk stripes is prioritized in HDFS [3] and WAS [7]. In such a manner, the repair time of high risk stripes is dramatically reduced. Meanwhile, the increase of the repair time is relatively small. Therefore, the data reliability and availability are improved. It is worth noting that, after being scheduled, the failure identification time becomes dominant in the repair time of high risk stripes, because those chunks in high risk stripes are usually very few. **As a result, the reduction in the identification time of high risk stripes is very effective in improving the data reliability and availability.**

Silberstein et al. propose a technique *Lazy* [14] to reduce the repair network traffic. Because chunks in low risk stripes, e.g., S^1, are dominant in all chunks be recovered, most of the repair network traffic is generated by recovering those chunks. Canceling the recovery of chunks in low risk stripes dramatically reduces the repair network traffic. However, the data reliability and availability dramatically decrease.

7 Conclusions

In this paper, we present a risk-aware failure identification scheme, named RAFI, to simultaneously improve the data reliability, availability, and serviceability (RAS) of erasure-coded data centers. The basic idea of RAFI is identifying a chunk failure not only according to its failure duration, but also based on the data reliability and availability of its host stripe. The benefits of RAFI are: (1) the identification of failed chunks in high risk stripes is expedited to improve the data reliability and availability; and (2) the identification of failed chunks in low risk stripes is postponed to reduce the repair network traffic, thus improving the serviceability. Our results based on both simulations and prototyping have demonstrated the effectiveness and efficiency of RAFI in terms of reduced data loss, unavailability, and repair network traffic.

8 Acknowledgment

We are grateful to anonymous reviewers and particularly our shepherd Dahlia Malkhi for their helpful comments and suggestions. We thank Changsheng Xie and Ting Ye for their great efforts on this paper. Shenggang Wan is the corresponding author. This research is sponsored by National Natural Science Foundation of China Grants Nos. 61300046 and 61331010, and U.S. National Science Foundation Grants Nos. CCF-1717660, CNS-1702474 and CCF-1547804.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *SOSP'03*, 2003.

[2] K. M. Greenan, X. Li, and J. J. Wylie, "Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," in *MSST'10*, 2010.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *MSST'10*, 2010.

[4] L. Xiang, Y. Xu, J. C. S. Lui, and Q. Chang, "Optimal recovery of single disk failure in RDP code storage systems," in *SIGMETRICS'10*, 2010.

[5] S. Li, Q. Cao, J. Huang, S. Wan, and C. Xie, "PDRS: A New Recovery Scheme Application for Vertical RAID-6 Code," in *NAS'11*, 2011.

[6] E. B. Nightingale, J. Elson, J. Fan, O. Hofmann, J. Howell, and Y. Suzue, "Flat Datacenter Storage," in *OSDI'12*. USENIX, 2012.

[7] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure Coding in Windows Azure Storage," in *ATC'12*, 2012.

[8] O. Khan, R. 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.

[9] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang, "NCCloud: applying network coding for the storage repair in a cloud-of-clouds," in *FAST'12*, 2012.

[10] I. Tamo, Z. Wang, and J. Bruck, "Zigzag Codes: MDS Array Codes With Optimal Rebuilding," *Transactions on Information Theory*, 2013.

[11] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing Elephants: Novel Erasure Codes for Big Data," in *VLDB'13*, 2013.

[12] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A Solution to the Network Challenges of Data Recovery in Erasure-coded Distributed Storage Systems: A Study on the Facebook Warehouse Cluster," in *HotStorage'13*, 2013.

[13] S. Xu, R. Li, P. P. C. Lee, Y. Zhu, L. Xiang, Y. Xu, and J. C. S. Lui, "Single Disk Failure Recovery for X-Code-Based Parallel Storage Systems," *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 995–1007, 2014.

[14] M. Silberstein, L. Ganesh, Y. Wang, L. Alvisi, and M. Dahlin, "Lazy Means Smart: Reducing Repair Bandwidth Costs in Erasure-coded Distributed Storage," in *SYSTOR'14*, 2014.

[15] K. V. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "Having your cake and eating it too: jointly optimal erasure codes for I/O, storage and network-bandwidth," in *FAST'15*, 2015.

[16] M. Xia, M. Saxena, M. Blaum, and D. A. Pease, "A Tale of Two Erasure Codes in HDFS," in *FAST'15*, 2015.

[17] L. Pamies-Juarez, F. Blagojević, R. Mateescu, C. Gyuot, E. E. Gad, and Z. Bandić, "Opening the Chrysalis: On the Real Repair Performance of MSR Codes," in *FAST'16*, 2016.

[18] S. Mitra, R. Panta, M. R. Ra, and S. Bagchi, "Partial-parallel-repair (PPR): a distributed technique for repairing erasure coded storage," in *EUROSYS'16*, 2016.

[19] R. Li, X. Li, P. P. C. Lee, and Q. Huang, "Repair Pipelining for Erasure-Coded Storage," in *ATC'17*, 2017.

[20] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in Globally Distributed Storage Systems," in *OSDI'10*, 2010.

[21] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly, "The quantcast file system," 2013.

[22] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Transactions on Information Theory*, 2010.

[23] D. S. Papailiopoulos, J. Luo, A. G. Dimakis, and C. Huang, "Simple regenerating codes: Network coding for cloud storage," in *INFOCOM'12*, 2012.

[24] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM'08*, 2008.

[25] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, and M. Zhu, "B4: experience with a globally-deployed software defined wan," in *SIGCOMM'13*, 2013.

[26] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, and P. Germano, "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," in *SIGCOMM'15*, 2015.

[27] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, 2007.

[28] AlibabaCloud, "Alibab ECS," https://www.alibabacloud.com/product/ecs, 2017.

[29] J. Fang, "DR-SIM," https://github.com/yydfjt/distributed_system_simulator, 2017.

[30] L. Wan, F. Wang, H. S. Oral, S. S. Vazhkudai, and Q. Cao, *A Report on Simulation-Driven Reliability and Failure Analysis of Large-Scale Storage Systems*, Nov 2014. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/1185665

[31] A. Oriani and I. C. Garcia, "From Backup to Hot Standby: High Availability for HDFS," in *SRDS'12*, 2012.

[32] A. Thomson and D. J. Abadi, "CalvinFS: Consistent WAN Replication and Scalable Metadata Management for Distributed File Systems," in *FAST'15*, 2015. [Online]. Available: https://www.usenix.org/conference/fast15/technical-sessions/presentation/thomson

[33] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast Crash Recovery in RAMCloud," in *SOSP'11*, 2011.

[34] B. gon Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient replica maintenance for distributed storage systems," in *NSDI'06*, 2006.

[35] V. Venkatesan, I. Iliadis, and R. Haas, "Reliability of Data Storage Systems under Network Rebuild Bandwidth Constraints," in *MASCOTS'12*, 2012.

[36] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in *SIGCOMM'12*, 2012.

[37] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the Social Network's (Datacenter) Network," in *SIGCOMM'15*, 2015.

[38] F. André, A.-M. Kermarrec, E. Le Merrer, N. Le Scouarnec, G. Straub, and A. Van Kempen, "Archiving cold data in warehouses with clustered network coding," in *EUROSYS'14*, 2014.

[39] S. Jiekak, A.-M. Kermarrec, N. Le Scouarnec, G. Straub, and A. Van Kempen, "Regenerating Codes: A System Perspective," in *SIGOPS'13*, 2013.

[40] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, 1960.

[41] A. K. Dutta and R. Hasan, "How much does storage really cost? Towards a full cost accounting model for data storage," in *International Conference on Grid Economics and Business Models*. Springer, 2013.

[42] Amazon, "Pricing of Amazon S3," https://aws.amazon.com/s3/pricing, 2017.