R-ADMAD: High Reliability Provision for Large-Scale De-duplication Archival Storage Systems

Chuanyi LIU[1], Yu GU[1], Linchun SUN[1], Bin YAN[1], Dongsheng WANG[1, 2]
1(Department of Computer Science and Technology, Tsinghua University, Beijing, China)
2(Research Institute of Information Technology, Tsinghua University, Beijing, China)
*Corresponding author: wds@tsinghua.edu.cn
(cy-liu04, gy98, slc, yanb)@mails.tsinghua.edu.cn;

ABSTRACT

Data de-duplication has become a commodity component in data-intensive systems and it is required that these systems provide high reliability comparable to others. Unfortunately, by storing duplicate data chunks just once, de-duped system improves storage utilization at cost of error resilience or reliability. In this paper, R-ADMAD, a high reliability provision mechanism is proposed. It packs variable-length data chunks into fixed sized objects, and exploits ECC codes to encode the objects and distributes them among the storage nodes in a redundancy group, which is dynamically generated according to current status and actual failure domains. Upon failures, R-ADMAD proposes a distributed and dynamic recovery process. Experimental results show that R-ADMAD can provide the same storage utilization as RAID-like schemes, but comparable reliability to replication based schemes with much more redundancy. The average recovery time of R-ADMAD based configurations is about 2-6 times less than RAID-like schemes. Moreover, R-ADMAD can provide dynamic load balancing even without the involvement of the overloaded storage nodes.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management; C.4 [**PERFORMANCE OF SYSTEMS**]: Reliability, availability;

General Terms

Reliability, Design, Algorithms, Measurement, Performance

Keywords

Data De-duplication, Error Correcting Code, Reliability

1. INTRODUCTION

The explosive growth of online digital content has resulted in enormous strain on storage systems [1]. As the volume of data grows, also increasing is the Total Cost of Ownership (TCO), which includes storage infrastructure cost, management cost and human administration cost. Therefore in large-scale distributed archival storage systems, reducing the amount of data that need to be transferred, stored, and managed becomes a crucial, and it also benefits for application performance, storage costs and administrative overheads [2].

As a result, Data De-duplication has received a broad attention in both academia and industry, and gradually it has become a commodity component in data-intensive storage systems and products (especially archival storage systems)[3], [4], [5], [6]. Data De-duplication refers to approaches that use lossless data compression schemes to minimize the duplicated data at inter-file level. It divides each file into a number of non-overlapping chunks and storing only unique chunks on storage devices.

However, as a lot of duplications are intended by users and applications for increased reliability or availability, especially in archival storage systems, data should be preserved over long time periods. This requires that the de-duplication storage systems provide reliability comparable to other high-available systems. Unfortunately, by storing duplicated data chunks just once, de-duped system achieves storage utilization at the cost of error resilience or having potential risk to reduce reliability. As dependencies are introduced between files that share the same chunks, if such a shared chunk is lost, a disproportionately large amount of data becomes inaccessible because of the unavailability of all the files that share this chunk. [8] (If the value of a chunk were measured in the amount of file data (user data) that would be lost on losing a single chunk, then in a traditional data store, each chunk would be equally valuable. While in a de-duplication based store, a chunk with a higher commonality would be more valuable.) Specifically, the amount of user data lost when a chunk i in the storage system is destroyed is given as follows [2]:

$$loss_i = commonality_i * chunksize \quad (1)$$

Based on the above analysis, we can see that guaranteeing high data reliability is a critical problem in de-duplicated storage systems. Although research on the reliability issues in storage systems is a hot topic and has been well-studied, reliability guarantee for de-duplication storage systems, especially in large-scale archival systems, does bring in some new challenges, which are summarized as follows:

- As reliability is generally achieved by redundancies, de-duplication and reliability provision, to a great extent, are intrinsically contradicting to one another. A natural way is to retain several copies of each chunk according to its value, as Bhagwat etc. [8] proposed. However, this approach requires a large amount of space and communication as the replication copy increases. So there should be reasonable tradeoff between utilization and reliability.

ICS'09, June 8-12, 2009, Yorktown Heights, New York, USA.

- Data chunks are usually variable sized after the chunking procedure, such as Rabin fingerprinting [22] and metadata aware chunking [7], while most of the reliability schemes work on fixed-size blocks (including XOR-based coding [37], Error Correcting coding (ECC) [24], [25], [26] and replication based schemes [32], [36]). Moreover, since the data chunks may be of different lengths, storing them in storage devices may result in a lot of fragments and involve a high percentage of random disk accesses, which is very inefficient. So how to exploit existing reliability algorithms so as to effectively and efficiently work on the variable data chunks should be dedicatedly devised, including data distribution and redundancy placement.

- As each data chunk may be shared by many files, load balance is also an important factor that should be considered when designing data distribution and recovery scheme. As redundancies are added, there is opportunity to leverage redundancy encoding to provide dynamic load balance at run-time according to time-variable hot spots.

Directed against the above challenges, this paper proposes R-ADMAD: a high-reliability provision mechanism for large-scale de-duplication archival storage systems. R-ADMAD divides the storage nodes into dynamic redundancy groups, which can also be configured by policy rules manually or generated by Metadata Server (MDS) according to current storage nodes status and actual failure domains. R-ADMAD packs de-duped variable-length data chunks into fixed sized objects, and exploit ECC codes, which can provide better fault tolerance than replication [36], to encode the objects and distributes them among the storage nodes in a redundancy group, within which a certain number of node failures can be tolerated through the distributed and dynamic recovery process proposed by R-ADMAD. Experimental results show that R-ADMAD can support archival performance and storage space utilization comparable with RAID-like systems, while at the same time with reliability and scalability comparable to replication based schemes with much more redundancies. For example, at some workloads a typical R-ADMAD configuration (*R-to100/168*) can provide the same storage utilization to RAID5 (*ra2/3*) scheme (with about 66.7% storage utilization), but comparable reliability with 3-way replication based scheme (with 33.3% storage utilization). Meanwhile, it is about 10% higher in reliability, 3 times less of average recovery time than RAID5 (*ra2/3*) scheme. Moreover, R-ADMAD can provide dynamic load balance even without the involvement of overloaded storage nodes.

2. RELATED WORK

Data de-duplication has become a commodity component in data-intensive storage systems and products. The Farsite [12] system may be the first study on exploring file-system existing commonality, but it works at file-level granularity. In order to eliminate the redundancies and similarities to the best extent, most de-dup schemes partition each file into a number of non-overlapping chunks and storing only unique chunks into storage devices. According to the chunk sizes and chunk boundaries, some corresponding algorithms have been used or proposed: CASPER [13] and Venti [39] adopt a fixed-sized file dividing method, and divide each file into predefined fixed-size chunks; LBFS [14] uses Rabin fingerprint algorithm to divide each file into variable sized chunks, and every chunk with a global unique identifier. Through comparing the identifiers, duplicate chunks can be eliminated; ADMAD [7] proposed an application-driven metadata aware de-duplication scheme, which used certain meta-data information at different levels of the I/O path to direct file partitioning into more meaningful data chunks, thereby further reducing the inter-file level duplications. There are also some schemes achieving de-duplication based on delta encoding [15], which generates the delta file given the source file (a.k.a reference file) and the target file. It is based on resemblance detection between data objects and uses delta encoding to store only deltas instead of entire data objects [16]. Recently, there is another method named Fuzzy Block Matching [17], whose idea originates from CASPER [13]. In this method, every object has some features [14], and through comparing the features of different objects, resemblance relationship can be determined.

By now there are also many commercial products and systems based on de-duplication techniques, such as Data Domain [4], Quantum Dxi Series [5], Centera [36] by EMC and PureDisk [6] by Symantec.

Unfortunately, de-dup systems gain storage utilization at the cost of decreased error resilience or increased potential of failures, which is not accepted in real deployment, especially in long-time archival storage systems. Bhagwat, et al. [8] first noticed this problem, and proposed a replication based approach, in which certain copies of each chunk are replicated on different devices according to their importance or popularity. This kind of approach is natural and easy to integrate into the de-duped storage systems, with a comparable availability to RAID-like systems (when the average number of copies of a chunk is bigger than three [8]), but it requires a large amount of space and communication, which is inefficient, especially when used in large-scale storage systems [18]. For example, writing large blocks to all replications is slow, space inefficient and difficult to manage. Moreover, how to actually organize the data chunks is not in the consideration of Bhagwat's work.

Another way is to directly use RAID system as the underlying storage. This kind of approach is transparent to upper layers and easy to deploy, but standard RAID systems can only tolerate up to two disk failures. Moreover, as the recovery process is relatively slow in RAID-like systems, the disk rebuild time lengthens as disk capacity grows [9]: When a TB disk fails, the rebuild usually can take upwards of 10 to 12 hours [11]. While hierarchical RAID or clustered RAID [45], [46] systems can tolerate multiple disk failures, and there are efforts which lead to scalable rebuild, RAID systems themselves are not scalable enough for nowadays large scale or massive (Petabyte-scale or Exabyte-scale) storage systems. RAID technology is no longer sufficient to guarantee the necessary high data reliability for large-scale storage systems [10].

In summary, designing a suitable mechanism with reliability guarantee particularly for large-scale de-duplicated storage systems is necessary.

3. R-ADMAD: PROVIDING HIGH RELIABILITY FOR DE-DUPLICATION SYSTEMS

R-ADMAD is mainly designed for large-scale disk-based online archival storage systems. It is distributed and scalable for large-scale archival or disaster recovery deployments. The storage pool is composed of heterogeneous Intelligent Storage Nodes (ISN), each of which consists of commodity elements such as off-the-shelf hardware, operating systems, file systems, and different types of storage sub-systems such as RAID, MAID [19] or directly attached SCSI, SATA or SAS disks can be equipped in ISN. The archival processes can be deployed on different application servers, and it is where the file chunking procedure takes place.

3.1 Architecture

The architecture of R-ADMAD is depicted in Figure 1. The system consists of three main components: Clients, Metadata Servers (MDS) and ISNs. Clients role as application servers in actual scenarios, where server side of applications such as email servers, multimedia servers, web servers, are deployed on them. File Archival Server (FAS), where file chunking is implemented, is also deployed on clients. The MDS takes charge of metadata management, token management and security management. It also makes decisions on chunk distribution and object redundancy placement by using its global knowledge. Data flow never goes through the MDS (chunks or objects are directly transferred between clients and ISNs) so as to minimize its involvement in reads and writes. ISNs are divided into ***redundancy groups***, each corresponding to a typical ECC configuration (redundancy groups can be dynamically generated at run-time). ISN exposes an object-based interface (OSD T-10[1]) [35] for data retrieval. On receiving the chunks from client, ISN packs the variable-sized chunks into fixed-sized objects, and uses ECC based redundancy schemes to encode and distribute these objects. Every group of objects (including the data objects and their ECC encoded parity/check objects) in a distribution operation is called an ***object group***, which is identified by *object_group_ID*. ISNs are based on commodity elements, and can be heterogeneous. They are interconnected with high speed storage area network.

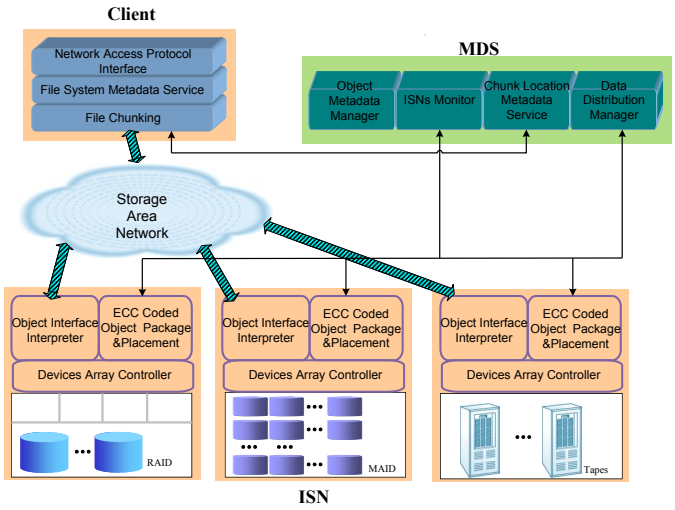


Figure 1. R-ADMAD Architecture for Large-scale De-duplication Archival Storage System

[1] OSD T10 standard has been ratified by ANSI in January 2005 as SCSI Object-Based Storage Device command interface extension.

3.1.1 Metadata Server (MDS)

To avoid detracting the main aspect of this paper, we mainly focus on the functions of namespace and metadata management for reliability mechanism. There are four important data structures in MDS:

(1) ***Redundancy_Group_Table***, which is used for redundancy group configurations. The size of a redundancy group can be varied according to different ECC schemes or different configuration parameters. The record structure for this table consists of the tuple < *group_ID, ISN_ID_list, ECC_scheme_ID, ECC_conf_parameters*>, where the *group_ID* is the identifier of the redundancy group, *ISN_ID_list* lists ISN identifiers in this redundancy group, *ECC_scheme_ID* is the identifier of ECC scheme for this group and *ECC_conf_parameters* is the configuration parameters for this ECC scheme.

(2) ***File_Attibutes_Table***, which is used to manage the file system namespace and to map hierarchical pathnames of regular files used by applications to chunk identifiers (from chunk identifiers we can further get mapping to the location of the object group that holds the very chunks). The record structure consists of tuple *<file_path, file_attributes_list, chunk_ID_list>*, where the *file_attributes_list* lists the attributes of the file used for metadata operations and index without actually fetching the file contents, *chunk_ID_list* lists all the chunk identifiers that belong to this file.

(3) ***Object_Group_Table***, which holds the metadata of object groups (the structure of an object stored on the ISN is shown in Figure 2). The record structure consists of tuple *<object_group_ID, group_ID, object_ ID_list>*, where the *object_group_ID* is the identifier of the object group, *group_ID* is the identifier of the redundancy group this object group belongs to, *object_ ID_list* lists the object identifiers in this object group.

(4) ***Chunk_Object_Table***, which is a Reverse Index [28] from *chunk_ID* to *object_group_ID* for locating the object a chunk belongs to.

3.1.2 Intelligent Storage Node (ISN)

ISN supports a scalable, efficient and self-manageable object-based storage model [32], [33], [34]. It sets up a shared environment which can both simplify administration work and reduce the number of distinct application systems. ISN is the foundation unit of storage servers, exposing an object-based storage interface, and an ISN physically consists of commodity components such as I/O channels, disk controllers, processors, and usually large amount of memory as well as network adapters.

ISN uses a flat namespace management, and every object is uniquely and globally identified by the object ID within the ISN. Object is the base unit of storage and access. Some prior research and practice [30] have demonstrated that using variable object size, especially isolating the real data, metadata, and attributes of an object into three independent files of underlying filesystems will not only severely hinder performance, but also dramatically increase the fragments within the storage devices, which will further slow down access [31]. Considering the characteristics of archival data workload, we adopt the fixed size object scheme, and the object size can be configured according to different applications and some related research and practice [32], [33]. The structure of an object is shown in Figure 2, where a chunk is the

minimal access unit for clients, specified by the chunk ID and the offset in an object (note that an object may contain part of, one or several chunks according to the size of the chunks).

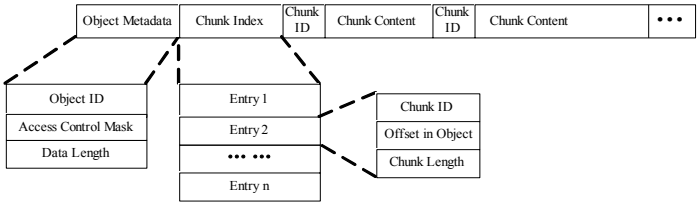


Figure 2. Structure of an object stored on ISN

ISN is an autonomous system, and its functions can be implemented as user space software daemons on top of the commodity Linux operation system.

3.2 Object Packing and ECC Based Distribution

The object packing and ECC based encoding and distribution procedure is illustrated in Figure 3.

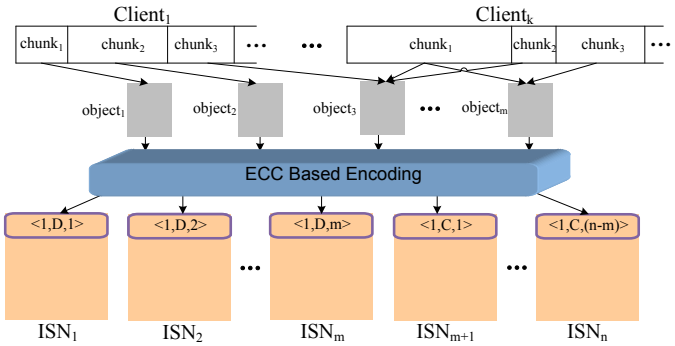


Figure 3. Object packing and ECC based object distribution procedure

To simplify the illustration, each object ultimately stored in ISN is marked as <*object_group_ID*, *data_or_check*, *object_ID*>, where *data_or_check* indicates whether the object is data object or check object. If there are new chunks to store, MDS tells which ISN the client should send these chunks to, and we call it Representative ISN (R-ISN for short). On receiving the R-ISN identifier from MDS, clients send the newly added chunk streams to the very R-ISN, where the chunks are packed into fixed-sized objects (e.g. *m* data objects). Data objects are fed into the ECC encoding module to generate *n* objects (including *(n-m)* check objects), and distributed into the ISNs among a redundancy group, preconfigured by MDS according to a pseudo-random hash function or manually determined policy rules [23]. For example, as ISNs with physical proximity, a shared power source, or a shared network can cause correlated device failures, also called Failure Domain [23]. To avoid this, we try to keep the ISNs in a redundancy group from different shelves, racks, power supplies, controllers, and/or physical locations.

Without losing generality, R-ADMAD chooses Reed–Solomon code [25] and Tornado code [26]. A unified API is exported by the ECC encoding & decoding module and listed in Table 1.

As every ISN in R-ADMAD can function as R-ISN, the ECC computation and data object distribution are totally distributed, which fully use both I/O and computing resources of ISNs. We can expect a reasonably good parallelism and integrated performance, which will be investigated in a separate paper.

3.3 Dynamic and Distributed Recovery

Inspired by the computation power and communication function embedded in ISN, R-ADMAD proposes a dynamic and distributed recovery. When an ISN failure has been detected, any ISN in the storage pool can be dynamically selected as the recovery target, and each object in the failed ISNs can be recovered in a distributed fashion by reading typical number of objects in the corresponding object group and error-correcting the failed object. As shown in Figure 4, when ISN_1 fails and objects *<1, D, 1>* and *<2, C, n>* are lost, we choose ISN_2 and ISN_m as the recovery target respectively. The target objects can be read in parallel from object group 1 and object group 2, and ECC decoding functions are performed in ISN_2 and ISN_m respectively to recover the lost objects. So with R-ADMAD, the recovery time can be largely reduced by parallelizing the data rebuild process, and the redundancies can be adaptively increased according to system load and capacity utilization. In other words, the window of vulnerability from the time needed to rebuild the data of failed ISNs to the time needed to create new redundancies is greatly reduced, as shown in Section 4.3.

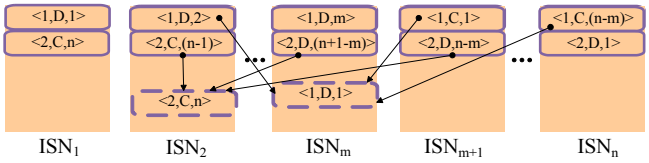
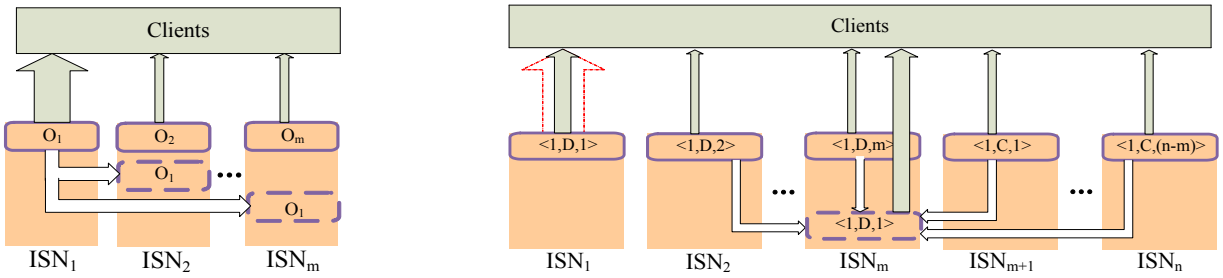


Figure 4. Recovery procedure upon ISN failure

Recovery mechanism in R-ADMAD can be further used for load balance, as shown in Figure 5. In traditional distributed storage systems, all data on heavily stressed storage server (hot spot) should be replicated or migrated to lighter ones in order to balance the workload(as illustrated in Figure 5(a)).

Table1. Unified Interface for ECC based Object Distribution and Recovery

API	Function Prototype
encode	status *st* = *encode*(object* *data_objs*, out object* *encoded_objs*);
Description: encode a series of data objects into encoded objects according to a certain redundancy group	
recover	status *st* = *recover*(object_id* *failed_objs_id*, object* *encoded_objs*, out object* *recovered_objs*);
Description: recover the failed objects given certain number of objects in the redundancy group	



(a) Load balance in a traditional distributed storage system **(b) Dynamic load balance with R-ADMAD**

Figure 5. Load balancing procedure with and without R-ADMAD

However, when a storage server is overloaded, its computation and I/O resources become so severe that the queuing effects cause the response time to increase exponentially. Therefore doing migration work under this condition would not increase the throughput of the whole system but make problem even worse.

In R-ADMAD, when an ISN is overloaded, another replica of all its data objects can be generated from other ISNs of the redundancy group by using ECC. After that, we can store generated data objects in relatively light ISNs and migrate some workload over to the light ISNs, as illustrated in Figure 5(b). The whole procedure does not add any extra tasks to the overloaded ISN, i.e. load balance is dynamic and totally transparent to the hot spots.

3.4 Archival & Retrieval Algorithm

Each file to be archived is first divided into a number of non-overlapping and variable-sized data chunks by using certain file chunking algorithms (e.g. Rabin fingerprinting [22], metadata aware chunking [7]). Each data chunk is identified by the hash value (chunk ID) generated by taking cryptographic hash functions, such as MD5, SHA1 [2], on their content. Then the metadata of these chunks are stored into MDS, where MDS determines whether there are duplicate chunks through comparing the chunk IDs. MDS also tells an R-ISN to which the client should send its chunks. The receiving R-ISN first packs received data chunks into fixed-sized objects, and then encodes and distributes them to ISNs within a redundancy group. The R-ISN sends the tuple < *group_ID, ISN_ID_list, object_ID_list* > to MDS after completing the objects distribution (the object write operation can be asynchronous if there is nonvolatile RAM equipped in ISN), where *group_ID* is the identifier of the redundancy group to which it belongs and *object_ID_list* is the identifier list of the objects responding to *ISN_ID_list* within the redundancy group. Finally MDS notifies the client of the completion of file archival if configured in synchronous mode. Clients can also cache the tuples of <*file path, offset, group_ID, object_ID_list*> to avoid the participation of MDS in further requests. The pseudo code of the archival algorithm is shown in Figure 6.

```
archive_file algorithm ()
//for Client
on receive (FILE) from applications
{ /* client get the file to be archived*/
    get the metadata information of the file, and store them into File_Attibutes_Table
    /* file chunking and chunk de-dup*/
    for each chunk in (chunks = fileChunking (file)) do
        if (chunk already exists) drop chunk;
        else (add chunk to chunk_stream);
    end for
    /*ask MDS for suitable R-ISN*/
    send message (ASK_FOR_R-ISN) to MDS
}
on receive (ISN_id) from MDS
{ /* client get the identifier of R-ISN*/
    Send (chunk_stream) to the ISN with id = ISN_id
}
//for MDS
on receive (ASK_FOR_R-ISN) from client
{/* determine a redundancy group by looking up from Group_Table or generating a new one*/
    group = generate_group ();
    /*select a R-ISN */
    ISN_id = select_R-ISN();
    send (ISN_id) to client
}
//for ISN
on receive (chunk_stream) from client
{
    pack the chunk_stream into m data objects {o1, o2 … om}
    encode(object* data_objs, object* encoded_objs), get the encoded n objects encoded_objs = {o1, o2 … om, c1,c2 … cn-m};
    send (objs[i]) to corresponding ISNi in the redundancy group
}
on receive (objs[i]) from R-ISN
{
    allocate object identifier for objs[i] and store it
    send (ACK(objs[i])) to R-ISN
}
End Algorithm
```

Figure 6. Description of the archival algorithm

When retrieving a file or reading part of a file content, the following three steps are performed: (1) Translate the offset and length from the point of file view into chunkIDs by looking up the

[2] Fortunately, the hash values for a specified hash function are inherently global unique and it is computationally infeasible to find two distinct inputs that hash to the same value [20].

File_Attibutes_Table in MDS; (2) look up ***Chunk_Object_Table*** to find the object group that stores the requested chunk (this operation is throughput sensitive and requires further optimization); (3) read relevant portion of the indicated object to buffer. Some of the attributes or metadata operations, such as key-word search, attributes retrieval, can directly operate on the MDS tables without the involvement of file contents stored as objects in ISNs.

4. EVALUATION

It is expensive and time consuming to build a petabyte-scale distributed storage system, and it is impractical for us to really run multi-year reliability test. Therefore an event-driven simulation is used to run the experiments. The simulator is based on PARSEC simulation tool [27].

The main objectives of the experiments include: (1) reliability comparisons of different schemes in metric of data availability ratio; (2) storage space utilization comparisons where utilization is defined as: *user data size / total occupied storage space * 100%*; (3) recovery performance comparisons in metric of average recovery time for an object (in RAID-like schemes, it is measured as the recovery time for identical data size as an object).

4.1 Experimental Setup

The system consists of 2000 ISNs, each with an extrapolated capacity of 1TB and an extrapolated sustainable bandwidth of 100MB/sec (based on reference of the current IBM enterprise storage specification [40]). We assume that recovery can use at most 20% of the available ISN bandwidth [9], and that each ISN reserves no more than 50% of its total capacity at system initialization for recovered data. We configure object size to 8MB, and object group (data objects plus check objects) size may vary according to different redundancy schemes.

Four types of redundancy schemes are evaluated: none, replication based (two-way mirroring (*rep1/2*), and three-way mirroring (*rep1/3*)), RAID5-like based (*ra2/3* and *ra4/5*), R-ADMAD with Reed Solomon coding (*R-rs4/6* and *R-rs8/10*) and with Tornado coding (*R-to48/96* and *R-to100/168*).

Since the datasets collected are much less than the data scale actually reside in real large scale storage systems, data amount allocated for each ISN is relatively very small, which makes the recovery too quick to mask the effect of ISN failures in real scenarios. Moreover, as the focus of our evaluation is to compare the relative efficiency of different schemes, we make two adjustments to solve the above problem: (1) augmenting the mathematical expectation of failure rate, but still making the rate curve conform to an industry standard for disk failure distribution [42], [43], with a typical End Of Design Life (EODL) set to 6 years; (2) scaling down recovery rate according to the data amount proportion between an ISN and a real world data server. Table 2 enumerates the ISN failure rates used in our experiments.

As the computation bandwidth in R-ADMAD (e.g. in Reed Solomon coding, it is mainly the Galois Fields [24], [25] arithmetic bandwidth, and in Tornado coding XOR operation bandwidth) is much faster than recovery rate [44], the computation cost is ignored in the simulation.

In our experiments, the de-duped data are initially distributed to ISNs randomly, and the redundancy group is also selected randomly from the active ISNs by MDS according to needed ISN number of different schemes. The system runs for 6 years, and with simulation granularity set to 200ms.

Table 2. ISN failure rates per 1000 hours

Period (month)	0-3	3-6	6-12	12-72
Failure rate	2%	1.4%	1%	0.8%

4.2 Workload Description

Four real-world datasets are used for evaluation listed in Table 3. *Log* is collected from 8 months access log from the apache server of http://learn.tsinghua.edu.cn. *Web* is collected by periodically crawling web pages from three similar websites within 10 months. *Backup* is a collection of daily archived personal working volumes in our research group for a month. *Gene-dbase* contains randomly chosen genomes from the NCBI GenBank database [41].

Table 3. Workload description

Name	Description	Size (G)	File #
Log	Log files of an apache server	250	2400
Web	Archives of some website contents	280	9M
Backup	A period of daily backups of personal volumes	3000	3K
gene-dbase	Collection of genes from GeneBank	93	500

4.3 Analysis of Results

We first measure time-variable data availability ratios for different workloads without any reliability mechanisms. As shown in Figure 7, data availability decrease un-proportionally with the lost ISNs over time. Especially with *backup* workload, the number of survival files drop rapidly to zero after just half year. This is because chunks in *backup* workload have higher commonality than other datasets, which leads to higher damage per loss of single chunk, and correspondly poorer error resilience. We think that there is relationship between the four workloads and the data availability. The average file sizes are 0.1GB, 30KB, 1GB, and 0.2GB for *log*, *web*, *backup* and *gene-dbase* respectively. For large file-sizes, the chunks have higher commonality and then higher damage per loss of single chunk, so the data availability drops more rapidly. It is converse for the small file-sizes.

Although different applications have different de-duplication characteristics, then have different effects upon storage failures, they do exhibit much poorer reliability than traditional storage systems, and thus require much more attention.

Figure 8 shows the reliability comparison of different schemes for 6 simulated years, measured every 6 months. In our experiments, the recovery mechanism in replication based schemes adopt the same mechanism proposed by R-ADMAD, i.e. the recovery is distributed, which will increase the reliability of replication based schemes. The data contents of an object group cannot be dynamically changed in replication based schemes, while R-

ADMAD can achieve this through ECC decoding as described in Section 3.3. From Figure 8 we observe that *rep1/3* and different configurations under *R-ADMAD* scheme generally have the best reliability, but space redundancy of *rep1/3* is much larger than R-ADMAD based ECC configurations. Reliability of *rep1/2* is poorer than R-ADMAD schemes with comparable redundancies (for *log* workload, *rep1/2* is 5.2% poorer than *R-to48/96*, and 4.9% poorer than *R-rs4/6*); Meanwhile, *R-rs8/10* and *ra4/5* have the worst reliability (e.g., for *log* workload, *ra2/3* is 4.4% poorer than *R-rs4/6*, and for gene-dbase workload, *ra4/5* is 8.2% poorer than *R-rs8/10*). This is mainly because the recovery process in RAID5-like systems is on ISN basis, and it cannot recover dynamically in distributed fashion as R-ADMAD does.

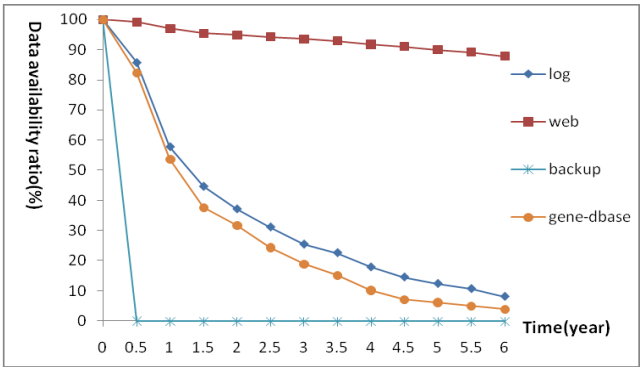
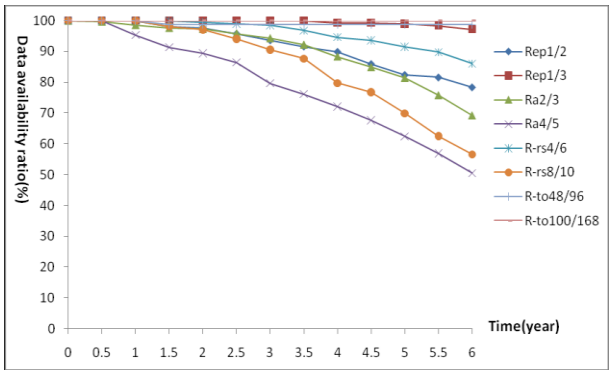
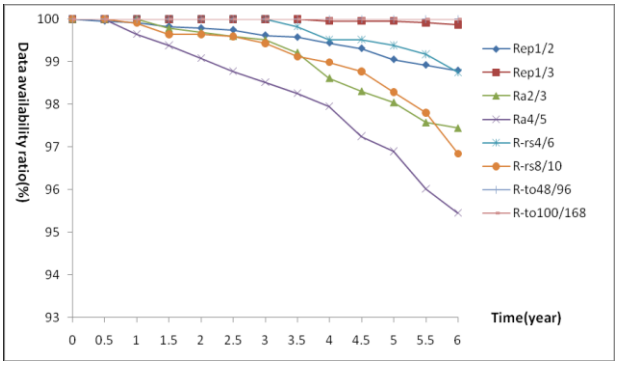


Figure 7. Data availability for different workloads without any reliability guarantee mechanisms

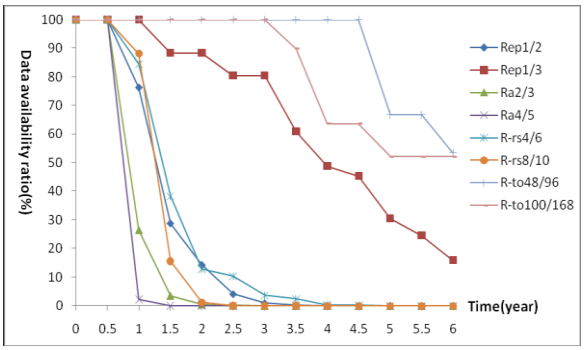


(a) log

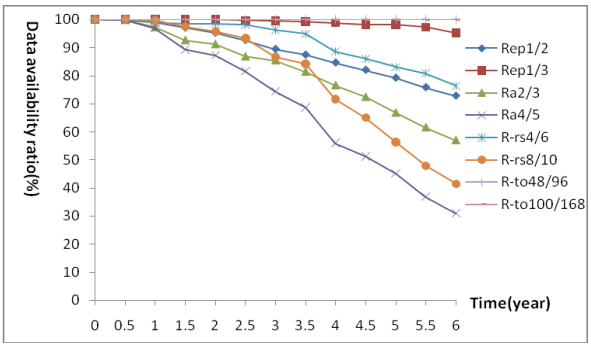


(b) web

By comparing the series in R-ADMAD we further observe that, reliability of *R-rs4/6* and *R-rs8/10* series are comparable with *R-to48/96* and *R-to100/168* series. For example, for *log* workload, the *R-rs4/6* is 1.6% worse than the *R–to100/168*. Reliability of *R-rs4/6* is better than *R-rs8/10,* e.g. for *log* workload, *R-rs4/6* is about 9.3% better than *R-rs8/10,* which mainly benefits from the increased redundancy and less data transmission when recovery procedure takes place. In contrast, reliability of *R-to48/96* is comparable with *R-to100/168* even with an increased redundancy. This is because Tornado coding is based on statistic distribution and recovery placement. For example, between 4.5 to 5 years of the *backup* workload, *R-to48/96* is even more reliable than *rep1/3* scheme. So we think Low Density Parity Check (LDPC) [26] codes like Tornado are suitable for large scale de-dup storage systems.



(c) backup



(d) gene-dbase

Figure 8. Data reliability comparisons with different workloads

To summarize, we can see that R-ADMAD based schemes provide much more reliability given similar space redundancies, i.e. R-ADMAD can provide good reliability with less redundancy.

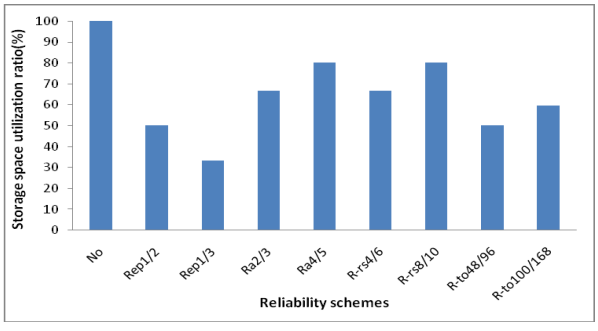


Figure 9. Storage utilization comparison of different reliability schemes

Figure 9 shows the storage utilization comparison for different reliability schemes, where *Ra4/5* and *R-rs8/10* have the highest utilization of 80%, and *Rep1/3* has the worst utilization of 33.3%. We can see that storage utilization of R-ADMAD is comparable with RAID-like systems. Moreover, it can be adaptively configured according to different application requirements.

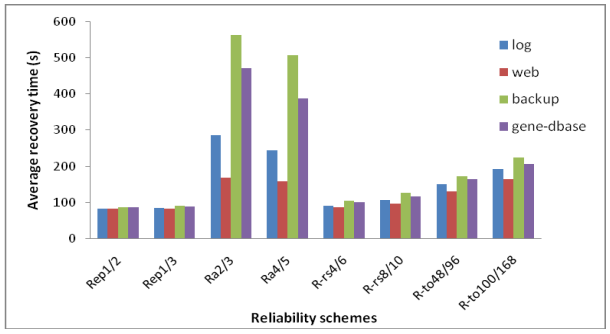


Figure 10. Recovery time comparison of different reliability schemes

Figure 10 shows the average recovery time for different schemes. We observe that *ra2/3* and *ra4/5* have much larger recovery time than others. For example, *ra2/3* is about 5 times bigger than rep1/2 with backup workload, and *ra4/5* is 2.5 times bigger than R-rs8/10 with log workload. This is mainly because the recovery of RAID5-like systems is on ISN basis, i.e. should an ISN fail, it would determine another recovery target from which to recover all the data on the original ISN. In contrast, the recovery of R-ADMAD is based on object, with a distributed and dynamic procedure, so it has little dependency on ISN capacity increase (while recovery time of RAID-like systems is sensitive to data capacity). Moreover, any ISN across the whole system can be the recovery target, so idle disk bandwidth may be used for recovery to achieve high efficiency.

5. CONCLUSION AND FUTURE WORK

Data de-duplication has become a commodity component for in data-intensive storage systems and products. This paper first explains the dramatically decrease of error resilience in de-duplication storage systems, and analyzes the challenges for providing high reliability in large scale de-duped systems. To address the challenges, this paper proposes R-ADMAD, with main contributions summarized as follows:

- To our best knowledge, R-ADMAD is the first attempt to design a high-reliability mechanism for de-duplication storage systems by exploiting ECC. R-ADMAD exhibits a good tradeoff between utilization and reliability, which is not possible for RAID-like schemes or replication based schemes.

- R-ADMAD proposes a mechanism to pack variable sized data chunks into fixed sized objects, which are convenient for redundancy schemes to work on. Moreover, by using object as storing unit, R-ADMAD can optimize access performance.

- R-ADMAD uses the commodity components to build Intelligent Storage Nodes (ISN), which are used as distributed storage servers. Based on ISN, R-ADMAD proposes dynamic and distributed recovery, which can greatly reduce the recovery time and dynamically balance workload.

Experimental results show that R-ADMAD can provide the same storage utilization as RAID-like schemes, but comparable reliability to replication based schemes with much more redundancies. As R-ADMAD proposes a distributed and dynamic recovery process, the average recovery time of R-ADMAD based configurations is about 2-5 times less than RAID-like schemes. Moreover, R-ADMAD can provide run-time dynamic load balance even without the involvement of the overloaded storage nodes.

In future, we will further evaluate the reliability, performance and recovery time of more LDAP codes under R-ADMAD mechanism, so as to provide quantitative comparisons for a wider variety of ECC coding choices. We also plan to construct an R-ADMAD prototype system and evaluate the performance and reliability under real environments. As every ISN in R-ADMAD can function as R-ISN, ECC computation and data object distribution are totally distributed, which can fully use both the I/O and computing resources of ISNs. A good parallelism and integrated performance is reasonably expected. Although it is difficult for us to deploy thousands of ISNs and run multi-year reliability test for evaluation, we plan to implement R-ADMAD as software daemons and deploy it in current p2p network.

6. ACKNOWLEDGMENTS

We thank our shepherd Darren Kerbyson and the anonymous reviewers for their helpful comments and advice. We also thank Qin Xin and Bo Mao for their feedback and discussion with us. This work is supported by the China NSFC Program under Grant No. 60833004 and No. 60673145.

7. REFERENCES

[1] J F Gantz, et al. The Expanding Digital Universe: A Forecast of Worldwide Information Growth through 2010. IDC, March 2007.

[2] Partho Nath; Bhuvan Urgaonkar; Anand Sivasubramaniam. Evaluating the usefulness of content addressable storage for high-performance data intensive applications. Proceedings of the 17th international symposium on High performance distributed computing, Boston, MA, USA, 2008

[3] EMC Centera. Content Addressed Storage. http://www.emc.com/pdf/products/centera/centera guide.pdf.

[4] Data Domain. http://www.datadomain.com.

[5] Quantum Dxi-Series. http://www.quantum.com/Products/

Disk-BasedBackup/index.aspx

[6] Symantec PureDisk. http://www.symantec.com/business/products/overview.jsp?pcid=2244&pvid=1381_1

[7] Chuanyi Liu, Yingping Lu, David Du and Dongsheng Wang. ADMAD: Application-Driven Metadata Aware De-duplication Archival Storage System. International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI 2008) held in conjunction with the 25th IEEE Conference on Mass Storage Systems and Technologies (MSST 2008)

[8] Deepavali Bhagwat, Kristal Pollack, Darrell D. E. Long, Thomas Schwarz, Ethan L. Miller, Providing High Reliability in a

Minimum Redundancy Archival Storage System, Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '06), September 2006, pages 413-421.

[9] Xin Qin, Miller E L, Schwarz T J E. Evaluation of Distributed Recovery in Large-scale Storage Systems. Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, 2004-06: 172-181.

[10] Qin Xin. Understanding and Coping with Failures in Large-Scale Storage Systems. Technical Report UCSC-SSRC-07-06, May 2007.

[11] David Reine. Enterprise Data Center Storage Issues. THE CLIPPER GROUP Navigator, September 11, 2008. Accessed from http://www.clipper.com/research/TCG2008043.pdf

[12] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed ?le system deployed on an existing set of desktop pcs. SIGMETRICS Perform. Eval. Rev., 28(1):34–43, 2000.

[13] N Tolia, M Kozuch, and M Satyanarayanan, et al. Opportunistic Use of Content Addressable Storage for Distributed File Systems. In Proc. of Usenix 2003 Annual Technical Conference, San Antonio, TX, USA

[14] Athicha Muthitacharoen, Benjie Chen, David Mazières, and Abhishek Rawat, A low bandwidth network file system. SOSP 2001.

[15] M. Ajtai, R. Burns, R. Fagin, D. D. E. Long, and L. Stockmeyer, Compactly encoding unstructured inputs with differential compression. Journal of the ACM, 49(3):318–367, May 2002.

[16] Lawrence L. You and Christos Karamanolis, Evaluation of Efficient Archival Storage Techniques. 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies. April 13-16, 2004, College Park, Maryland, USA

[17] Bo Han and Pete, Keleher.Implementation and Performance Evaluation of Fuzzy File Block Matching. 2007 USENIX Annual Technical Conference, Santa Clara, CA, June 17–22

[18] N. Spillers. Storage Challenges in the Medical Industry. In The 4th Intelligent Storage Workshop, Digital Technology Center, University of Minnesota, 2006.

[19] Dennis Colarelli, Dirk Grunwald. Massive arrays of idle disks for storage archives. Proceedings of the 2002 ACM/IEEE conference on Supercomputing, p.1-11, November 16, 2002, Baltimore, Maryland

[20] B Van Rompay, On the security of dedicated hash functions. In the 19th Symposium on Information Theory in the Benelux, 1998

[21] Xin Qin, Miller E L, Schwarz T J E. Evaluation of Distributed Recovery in Large-scale Storage Systems. Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, 2004-06: 172-181.

[22] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[23] Sage Weil, Scott A. Brandt, Ethan L. Miller, Carlos Maltzahn, CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data, Proceedings of SC '06, November 2006.

[24] W.W. Peterson and E. J.Weldon, Jr., Error-Correcting Codes, Second Edition. MIT Press, Cambridge, MA, 1972.

[25] F. J. MacWilliams and N. J. A. Sloane. The Theory of Error-Correcting Codes, Part I. North-Holland, Amsterdam, 1977

[26] Luby, M. G., M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman, “Efficient Erasure Correcting Codes”, IEEE Transactions on Information Theory, 47(2), 569-584, February 2001.

[27] R. A. Meyer and R. Bagrodia. PARSEC user manual, release 1.1. http://pcl.cs.ucla.edu/projects/parsec/.

[28] S. Brin and L. Page.,The anatomy of a large-scale hypertextual web search engine. In WWW Conference, volume 7, 1998.

[29] MySQL. http://www.mysql.com.

[30] David Du, Dingshan He, Changjin Hong, Jaehoon Jeong, Vishal Kher, Yongdae Kim, Yingping Lu, Aravindan Raghuveer, and Sarah Sharafkandi, "Experiences in Building an Object-Based Storage System based on the OSD T-10 Standard," Submitted to 14th NASA Goddard & 23rd IEEE (MSST2006) Conference on Mass Storage Systems and Technologies May 15–18, 2006, College Park, MD

[31] Mendel Rosenblum and John K. Ousterhout, The design and implementation of a log-structured file system. ACM Transactions on Computer Systems (TOCS), 10(1), February 1992.

[32] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. Proc. SOSP’03, 2003.

[33] Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn, Ceph: A Scalable, High-Performance Distributed File System, Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06), November 2006.

[34] Lustre Object-based Cluster File System. http://www.sun.com/software/products/lustre/index.xml

[35] Storage Networking Solutions. Object Storage Architecture: Defining a new generation of storage systems built on distributed, intelligent storage devices. http:// www.snseurope.com/featuresfull.php?id=2193. 2004, 9

[36] Weatherspoon, H. and J. Kubiatowicz, “Erasure Coding vs. Replication: A Quantitative Comparison”, Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002), March 2002, 328-338.

[37] J. J.Wylie and R. Swaminathan. Determining fault tolerance of XOR-based erasure codes efficiently. In DSN-2007, pages 206–215. IEEE, June 2007.

[38] The Zebra Striped Network File System, John Hartman and John Ousterhout, ACM TOCS 1995.

[39] S Quinlan and S Dorward. Venti: A New Approach to Archival Storage. In Proceedings of Conference on File and Storage Technologies (2002).

[40] IBM Enterprise disk storage. http://www.ibm.com/systems/ storage/disk/enterprise/ds_family.html

[41] NCBI GenBank. http://www.ncbi.nlm.nih.gov/Genbank/.

[42] J. G. Elerath. Specifying reliability in the disk drive industry: No more MTBF's. In Proceedings of the 2000 Annual Reliability and Maintainability, pages 194–199. IEEE,2000.

[43] Bianca Schroeder; Garth A. Gibson. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? 5th USENIX Conference on File and Storage Technologies (FAST 2007)

[44] James S. Plank, Jianqiang Luo, Catherine D. Schuman, Lihao Xu, Zooko Wilcox-O'Hearn. A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage. In Proceedings of the Seventh USENIX Conference on File and Storage Technologies (FAST) 2009, San Francisco, CA

[45] Sung Hoon Baek, Bong Wan Kim, Eui Joung Joung, Chong Won Park. Reliability and performance of hierarchical RAID with multiple controllers. Proceedings of the twentieth annual ACM symposium on Principles of distributed computing , Newport, Rhode Island, United States, Pages: 246 – 254, 2001

[46] Kai Hwang, Hai Jin, Roy Ho. RAID-x: A New Distributed Disk Array for I/O-Centric Cluster Computing. In proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, PA, PP279—287, 2000.