ERMS : An Elastic Replication Management System for HDFS

Zhendong Cheng, Zhongzhi Luan, You Meng, Yijing Xu, Depei Qian
Sino-German Joint Software Institute
Beihang University, Beijing, China
{zhendong.cheng, zhongzhi.luan, you.meng, yijing.xu depei.qian}
@jsi.buaa.edu.cn

Alain Roy
Computer Sciences Department
University of Wisconsin-Madison
Madison, USA
roy@cs.wisc.edu

Ning Zhang, Gang Guan
Tencent Research
Shenzhen, China
{ningzhang, gangguan}
@tencent.com

***Abstract*—The Hadoop Distributed File System (HDFS) is a distributed storage system that stores large-scale data sets reliably and streams those data sets to applications at high bandwidth. HDFS provides high performance, reliability and availability by replicating data, typically three copies of every data. The data in HDFS changes in popularity over time. To get better performance and higher disk utilization, the replication policy of HDFS should be elastic and adapt to data popularity. In this paper, we describe ERMS, an elastic replication management system for HDFS. ERMS provides an active/standby storage model for HDFS. It utilizes a complex event processing engine to distinguish real-time data types, and then dynamically increases extra replicas for hot data, cleans up these extra replicas when the data cool down, and uses erasure codes for cold data. ERMS also introduces a replica placement strategy for the extra replicas of hot data and erasure coding parities. The experiments show that ERMS effectively improves the reliability and performance of HDFS and reduce storage overhead.**

Keywords: HDFS; Elastic; Replication Management

I. INTRODUCTION

The storage demands of cloud computing have been growing exponentially year after year. Rather than relying on traditional central large storage arrays, the storage system for cloud computing consolidates large numbers of distributed commodity computers into a single storage pool, and provides a large capacity and high performance storage service in an unreliable and dynamic network environment at low cost. To build such a cloud storage system, an increasing number of companies and academic institutions have started to rely on the Hadoop Distributed File System (HDFS) [1]. HDFS provides reliable storage and high throughput access to application data. It is suitable for applications that have large data sets, typically the Map/Reduce programming framework [2] for data-intensive computing. HDFS has been widely used and become a common storage appliance for cloud computing.

Based on data access patterns and data popularity, the data in HDFS could be classified into four types: hot data, cooled data, cold data and normal data. Hot data is the popular data, which means the data receives not only a large number of concurrent accesses, but also a high intensity of access. Cooled data is the no longer heavily accessed hot data. The unpopular data that is rarely accessible is cold data. The rest belongs to normal data. Normally, data changes in popularity over time, so the data types also change dynamically. Their popularity spikes when the data is freshest and decays as time goes by. The typical lifecycle of a data is as follows: After being created, the data becomes hot data because there are many requests for it. When the requests are completed, it becomes cooled. And then, the data is normal data. If they are rare accessed, the data turns cold.

Placing data as close as possible to computation is a common practice of data-intensive systems, which is called data locality. Data replication has been widely used as a means of providing high performance, reliability and availability of large-scale cloud storage systems, where failure is the norm rather than the exception. The management of replicas is critical to HDFS reliability and performance of HDFS.

HDFS uses an uniform triplication policy (i.e. three replicas for each file) to improve data locality and ensure data availability and fault tolerance in the event of data and disk failures. This policy could also achieve load balancing by distributing work across the replicas. For the common case, the triplication policy in HDFS works well in term of high reliability and high performance. However, there are two problems with this policy.

First, in a large and busy HDFS cluster, the hot data could be requested by many distributed clients concurrently. Replicating the hot data only on three different nodes is not enough to avoid contention for the datanodes which store hot data. If the number of jobs concurrently accessing hot data exceeds the number of replicas, some of these jobs may have to access data remotely and compete for the same replica.

Second, the triplication policy comes with a high overhead cost in terms of management for the cold data. Too many replicas may not significantly improve availability, but bring unnecessary expenditure instead. The management cost, including storage and network bandwidth, will significantly increase with the high number of replica.

Therefore the hot data should be assigned with a larger number of replicas to improve data locality of concurrent accesses. The additional copies may be used to improve not only availability, but also provide load balanceing and improve overall performance if replicas and data accessing requests are reasonably distributed. On the other hand, the overheads of cold data lead us to reconsider erasure codes as

978-0-7695-4844-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ClusterW.2012.25

IEEE computer society

an alternative for the cold data. Erasure code, which is a form of software RAID, aims to provide a highly reliable storage with lower disk cost. The performance and management cost tradeoffs between replication and erasure coding are well understood and have been evaluated in many environments. Erasure codes could be used in HDFS to provide potential storage and network savings to replication.

In view of these issues, we design and implement ERMS, an elastic replica management system for HDFS. ERMS introduces an active/standby storage model, takes advantage of a high-performance complex event processing (CEP) [3] engine to distinguish the real-time data types, and provides an elastic replication policy for the different types of data. ERMS uses Condor [4] to increase the replication number for hot data in standby nodes, and to remove the extra replicas after the data cooling down. The erasure codes could be used to save storage space and network bandwidth when the data becomes cold.

The contributions of this paper are: (1) introducing an Active/Standby storage model for data storage in HDFS cluster; (2) bringing in a new method, complex event processing, to distinguish data types in real-time; (3) presenting the replica placement strategy which is more useful than the original one; (4) based on these method, designing, implementing, and evaluating an elastic replica management system to manage the replica in HDFS.

The remainder of this paper is organized as follows. Section II provides an overview of background and discusses related work. The design of Elastic Replication Management System for HDFS (ERMS) is presented in Section III. We evaluate system matrices of HDFS and experiment ERMS in Section IV. Conclusions and possible future work are summarized in Section V.

II. Background and Related Work

HDFS is designed for reliably storing very large files across distributed commodity machines in a large cluster. [5] It stores each file as a sequence of blocks; all blocks in a file are of the same size, except the last one. The blocks of a file are replicated for reading performance and fault tolerance. HDFS introduces a simple but highly effective triplication policy to allocate replicas for a block. The default replica placement policy of HDFS is to put one replica on one node in the local rack; another on a node in a remote rack; and the last on a different node in the same remote rack. This replica placement policy cuts the inter-rack write traffic, which generally improves writing performance. The purpose of the rack-aware replica placement strategy is to improve data reliability, availability, and network bandwidth utilization. Administrators of HDFS clusters can specify the default replication factor (number of replicas) or the replication factor for single data. They can also implement their own replica placement strategy for HDFS.

Replication factor and replica placement are key issues of replication management. The issue of dynamic replication management strategy for HDFS has drawn considerable attention.

CDRM [6] is a cost-effective dynamic replication management scheme for large-scale cloud storage system. It builds up a cost model to capture the relationship between availability and replication factor. Based on this model, lower bound on replica reference number to satisfy availability requirement can be determined. CDRM further places replicas among distribute nodes to minimize blocking probability, so as to improve load balance and overall performance.

Scarlett [7] is an off-line system that replicates blocks based on the observed probability in a previous epoch. Scarlett computes a replication factor for each file and creates budget-limited replicas distributed among the cluster, with the goal of minimizing hotspots. Replicas are aged to reserve space for new replicas.

DARE [8] is an adaptive data replication mechanism for HDFS. It uses probabilistic sampling and a competitive aging algorithm independently at each node to determine the number of replicas to allocate to each file and the location to each replica. DARE takes advantage of existing remote data retrievals and selects a subset of the data to be inserted into the file system, hence creating a replica without consuming extra network and computation resources.

There also has been some work on erasure codes for HDFS.

DiskReduce [9] is a modification of the HDFS that enables asynchronous encoding of triple replicated data and provides RAID-class redundancy overheads. In addition to increasing a cluster's storage capacity as seen by its users with up to three factors, DiskReduce can delay encoding long enough to deliver the performance benefits of multiple data copies.

Khan et al. [10] presents an algorithm that finds the optimal number of codeword symbols needed for recovery for any XOR-based erasure code and produces recovery schedules that use a minimum amount of data. This algorithm improves I/O performance in practice for the large block sizes used in cloud file systems, such as HDFS.

CDRM, Scarlett and DARE try to figure out suitable replication factor for every file and place them in reasonable datanodes according to the current workload and node capacity. However, they do not consider the availability of data that has low replica factors. DiskReduce introduces a RAID policy for seldom used data, but does not point out how to judge the cold data. In ERMS, we use different replica policies for different data. For hot data it increases replication number to improve performance. For cold data it uses RAID to save storage space. For normal data it uses the default triplication policy. We also describe a specific replica placement strategy for the extra replicas of hot data and RAID parity.

III. Elastic Replication Management System

The purpose of this paper is to design an elastic replica management system for HDFS that seeks to increase data locality by replicating the hot data while keeping a minimum number of replicas for the cold data and encoding them. Additionally, ERMS dynamically adaptes to the changes in data access patterns and data popularity. In this section, we first describe the architecture of ERMS, and then we

introduce the active/standby storage model, data judging policy and replica placement strategy used by ERMS.

A. System Architecture

The architecture of ERMS is shown in Fig. 1. It automatically manages the replication number and replica placement strategy in HDFS clusters. HDFS is the basic storage appliance. The Data Judge Module obtains system metrics from HDFS clusters and uses CEP to distinguish current data types in real-time. According to the different types of data, the manager of ERMS could schedule replication manager tool and erasure coding tool to manage the replicas of data. Condor is an appropriate choice because it implements flexible scheduling system for distributing computing.

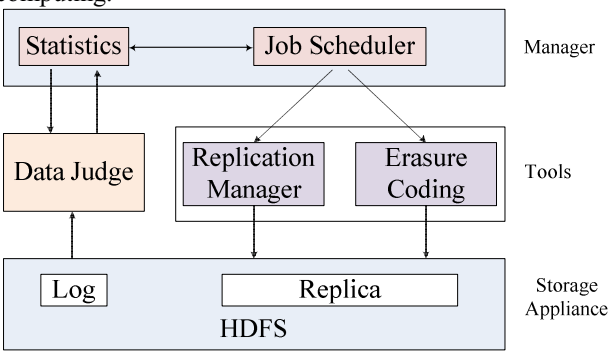


Figure 1. System Architecture of ERMS

The CEP engine delivers high-speed processing of the different events, identifies the most meaningful events from event clouds, analyzes their correlation, and takes action in real time. [3] It has increasingly become a choice for applications that require high-volume, low-latency and complex event correlation processing. The sliding window, typically the time window and length window, is one of the major features of CEP systems. The length window instructs the system to only keep the last N events. The time window enables us to limit the number of events within a specified time interval. ERMS makes use of CEP to analyze HDFS audit logs and distinguish real-time data types in HDFS.

Condor, which provides mechanisms and policies that support High Throughput Computing on large collections of distributed computing resources, can be used to manage the replication management system. Condor ClassAds is a flexible mechanism for representing the characteristics and constraints of nodes and replicas. The ClassAds mechanism is used in ERMS to detect when datanodes are commissioned or decommissioned in the cluster, and to judge whether the replicas are added or removed successfully. In addition, Condor is a well-functioning scheduling system. It schedules the increasing replication tasks and erasure decoding tasks immediately, while run the decreasing replication tasks and erasure encoding tasks when the HDFS cluster is idle.

To enhance the reliability of ERMS, the Condor log mechanism is used to record all replication manager tasks and erasure coding tasks. If these tasks failed, they could rollback automatically. We can replay all operations and analyze them.

B. Active/Standby Storage Model

ERMS introduces an active/standby storage model for HDFS cluster, as shown in Fig. 2. This model classifies the storage nodes into two types: active nodes and standby nodes. By keeping all nodes active, the additional nodes may be used to improve not only availability, but also provide load balancing and improve overall performance. However, the availability and performance increases were limited. In addition, this causes increased energy consumption, a significant problem for data centers. Therefore we use an active/standby storage model instead of keeping all nodes active.

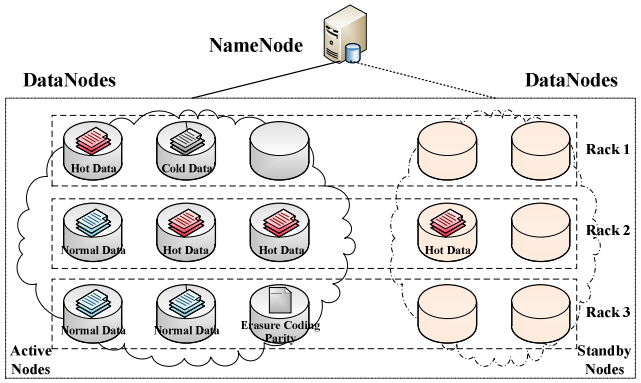


Figure 2. Active/Standby Storage Model

In HDFS clusters, nodes are placed in different racks. In the Active/Standby Storage Model, the active nodes and standby nodes are both distributed in different racks to make full use of data locations.

In a large and busy HDFS cluster, active nodes could be busy. In order to handle suddenly increasing access requests for hot data, ERMS commissions standby nodes and place the extra replicas at these nodes. Standby nodes might be better than active nodes when the active nodes are heavily used.

In addition, when decreasing the extra replicas of hot data, ERMS does not need to rebalance the replicas if they are located in standby nodes. This is because the data statuses of running nodes are not changing. It is desirable to avoid rebalancing because it takes considerable time and bandwidth. After all data in a standby node are removed, ERMS could shut down that node for energy saving.

C. Data Judge

Replication factor is a key issue of replication management in HDFS. The Data Judge Module utilizes CEP to distinguish real-time data types, and then determine the optimize replication factor for every data.

In a HDFS cluster, there are p active datanodes (DN_{A1}, DN_{A2}, … , DN_{Ap}) and q standby datanodes (DN_{S1}, DN_{S2}, …, DN_{Sq}). The block size of HDFS is S_B and the default replication factor is r_D ($0 < r_D < p$).

For the data D, the data size is S_d. So the block number n_d equals S_d/S_B (B_{D1}, B_{D2}, … , B_{Dnd}). The current replication factor of D is r ($0 < r < p+q$).

We developed a log parser to analyze the HDFS audit logs and translate the logs records into events for CEP system. A feature of CEP system is the ability to processing large volumes of messages and reacting in real-time to conditions that occur. It makes CEP suitable for analyzing the audit logs of HDFS, which are always very large and increase quickly. Usually, CEP system uses an SQL-standard-based continuous query language to express the query demands. Taking advantage of the time window t_w of CEP, ERMS could obtain concurrent access number of every data and block within the time t_w. We use N_d and N_b to represent the access number of data D and block B.

The experiments show that the number of concurrent access is a sufficient metric to capture the current type of data.

Hot data is the data that receives not only a large number of accesses, but also a high intensity of access. Three aspects could reveal that the data is hot. First, if the average concurrent access requests on every replica are high, the data is sure to be hot data. As the Formula (1), τ_M stands for the threshold, which is the largest access number one data replica could be hold. Second, data access could be locality. The access intensity of one block may be high while the concurrent access of the total data is regular (Formula (2)). M_M is the maximum access number of one block. Third, the data is hot when there are suitable presences of blocks being intensely accessed. In the Formula (3), M_m is the suitable access number of one block ($M_m < M_M$) and ε is the optimal percentage to measure the hot data.

$$N_d / r > \tau_M \quad (1)$$

$$\exists i \in [1, n_d], N_{b_i} / r > M_M \quad (2)$$

$$\forall j \in [1, n_d], count\left(N_{b_j} / r > M_m\right) \Big/ n_d > \varepsilon \quad (0 < \varepsilon < 1) \quad (3)$$

A datanode can simultaneously support a limited number of sessions due to capacity constraint, which is when the number of sessions has reached its upper bound, and the connection requests from application servers will be blocked, or rejected. For datanode DN, DN contains blocks B1, B2, … , and Bn. Formula (4) shows that the total number of data access on DN ($\sum_{i=0}^{n} N_{b_i} / r_{b_i}$) is larger than the threshold τ_{DN}. In this case, ERMS could choose the data D that contributes the largest access to DN and increased the replication factor of D.

$$\sum_{i=0}^{n} N_{b_i} / r_{b_i} > \tau_{DN} \quad (4)$$

The method to distinguish between cooled data is simple. Cooled data is the lightly accessed hot data. So the number of concurrent accesses of them are low, as the Formula (5) shown.

$$N_d / r < \tau_d \quad (5)$$

Cold data is the rarely accessed data, which means the last access time of the data is oldand the access number is low. Formula (6) shows the feature of cold data ($0 < \tau_m < \tau_d < \tau_M$), T_n stands for the current time and T_a stands for the last access time.

$$N_d / r < \tau_m \,\&\&\, T_n - T_a > t \quad (6)$$

The thresholds, M_M, M_m, τ_M , τ_m , τ_d , and τ_{DN} , are influenced by the HDFS cluster environments, which includes the types of disks, network bandwidth, CPU speed, etc. ERMS could dynamically change these thresholds based on system environments.

Different replication strategy could be used for different type of data after the data type having been determined. For the hot data, the ERMS can start standby nodes and increase replicas on standby nodes. It could remove the extra replicas of cooled data and shut down standby nodes if necessary. After the data becoming cold data, erasure codes are used to save storage space.

D. Replica Placement

The replica placement is critical to reliability and performance of HDFS. Ford et al. [14] characterized the availability properties of cloud storage systems based on an extensive one year study of Google's main storage infrastructure and presented statistical model. Their analysis concluded that data placement strategies need to be aware of failure groupings and failure bursts. The purpose of the default rack-aware replica placement strategy in HDFS is to improve data reliability, availability, and network bandwidth utilization.

ERMS introduces an Active/Standby storage model and uses different replication strategies for different types of data. The extra replicas of hot data are mainly used for data availability and performance, while the erasure coding parities are created for data reliability. So it needs a special replica placement strategy to take full advantage of the Active/Standby storage model and the features of data.

Algorithm 1: The Replica Placement Algorithm of ERMS

```
Active Datanodes DN_A←{DN_A1, DN_A2, … , DN_Ap}
Standby Datanodes DN_S←{DN_S1, DN_S2, …, DN_Sq}
Default replication factor r_D (0 < r_D < p)
Current replica number r (0 < r < p+q)
Block B ∈ Data D

Choose Datanode DN to place B
if B = Coding Block
  for DN_Ai in DN_A
    if DN_Ai contains the smallest number of D's blocks
      DN ← DN_Ai
      return DN
      end if
  end for
else if B = Data Block
  if r < r_D
    for DN_Ai in DNA
      if DN_Ai is suitable for the default replica placement strategy
        DN ← DN_Ai
```

```
19        return DN
20      end if
21    end for
22   else if r >= r_D
23    for DN_Si in DN_S
24      if DN_Si doesn't contain B
25        DN ← DN_Si
26        return DN
27      end if
28    end for
29    if DN = NULL
30      for DN_Ai in DN_A
31        if DN_Ai doesn't contain B
32          DN ← DN_Ai
33          return DN
34        end if
35      end for
36    end if
37   end if
38 end if

39 Choose Datanode DN to delete B
40 for DN_Si in DN_S
41   if DN_Si contains B
42     DN = DN_Si
43     Return DN
44   end if
45 end for
46 for DN_Ai in DN_A
47   if DN_Ai contains B
48     DN = DN_Ai
49     return DN
50   end if
51 end for
```

In light of this problem, we implement a pluggable replica placement strategy for HDFS, as shown in Algorithm 1. There are two types of block in ERMS: the data block and erasure codes parities. The block is an extra block of hot data if the replica factor of the block is no less than default replica factor. When choosing a datanode for data block, ERMS prefers to choose a standby node that is placed in the same racks with the other replica of the block for the extra block. It would use the default replica placement strategy for the normal data block. For the erasure codes parity, it could select the active node that contains the minimum number of data block of the same data. When deleting blocks, ERMS could prefer to delete them from standby nodes. In this replica placement strategy, it does not need to re-balance when increasing and decreasing the replication factor. If the erasure codes parities are located in the same nodes with the original data, the data will be lost and could not be recovered if these nodes are crashed. This strategy enhances the data availability.

IV. Experimental Testing and Analysis

A. Experiment Environment

We evaluated ERMS in a private cluster with one namenode and eighteen datanodes, all commodity computers. The namenode has two Intel Xeon E5520 CPUs of 2.26GHz, 12GB memory, 250GB SATA disk. The operating system is CentOS 5.5 with kernel 2.6.18-194.el5. The datanodes have an Intel Xeon E5420 CPU of 2.50GHz, 8GB memory, 60GB or 250GB SATA disk. The operating system is CentOS 5.4 with kernel 2.6.18-194.32.1.el5. The Java version is 1.6.0_24. These nodes locate in three different racks with Gigabit Ethernet network connecting.

The log parser for HDFS audit logs is developed in Java and we use CEP engine to analyze these logs. The total code is 2186 lines. We implement the ERMS in Hadoop-20, which is Facebook's realtime distributed Hadoop based on Apache Hadoop 0.20-append [1]. We have to modify the replica placement mechanism and add configuration parameters (active/standby nodes) to suit the ERMS.

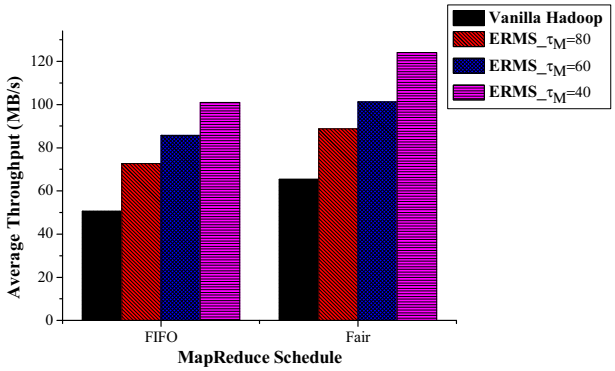
B. Performance and Analysis

We run jobs synthesized from the Statistical Workload Injector for MapReduce (SWIM) [17], which provides a one mouth job trace and replay scripts of a Facebook 3000-machine production cluster trace.

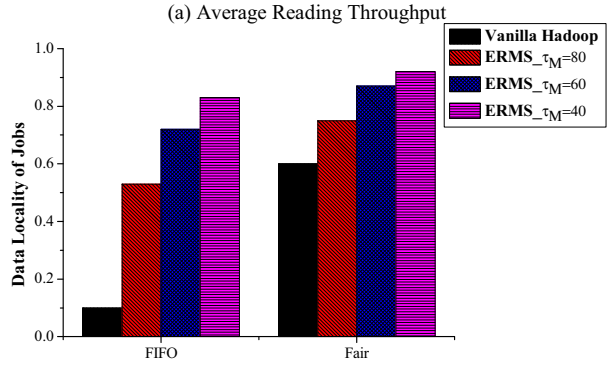
ERMS is scheduler independent, but different schedulers have different performance when running tasks. Therefore we evaluated it using FIFO scheduler and the Fair scheduler under different thresholds.

Reading throughput and data locality are two critical metrics for performance of HDFS. Reading throughput directly reflects system performance of file system. Data locality could reduce pressure on the network fabric, which is desirable in data-intensive scenarios since network fabrics are frequently oversubscribed. Fig. 3(a) and Fig. 3(b) show that ERMS could effectively improve reading throughput and data locality. It improves 50% – 100% reading throughput and 5 – 8 times data locality for FIFO scheduler, and 40% – 100% reading throughput and 20% – 70% data locality for Fair scheduler. The Fair scheduler is able to increase data locality at the cost of a small delay for tasks. Even in this case ERMS is able to increase locality.

The thresholds, M_M, τ_M, and τ_{DN}, are important parameters for ERMS. It is a tradeoff between system performance and storage cost. We can get high performance with a high overhead cost if these thresholds are low.



(a) Average Reading Throughput



(b) Data Locality

Figure 3. Reading Performance and Data Locality of ERMS

The cumulative distribution function of the data at the time they are accessed is shown in Fig. 4. It shows the data access patterns of the HDFS cluster. Fig. 5 is system storage space utilization during the experiments. It matches the Cumulative Distribution Function (CDF) of data access number. ERMS increases replica number of hot data, so the storage space is larger than normal when data access is heavily. For the cold data, which concurrent accesses number τ is lower than τ_m, ERMS uses Reed Solomon codes to encode it, with a replication factor of one and four coding parities. The results show that this erasure codes could significantly reduce storage overhead and doesn't heart data reliability.

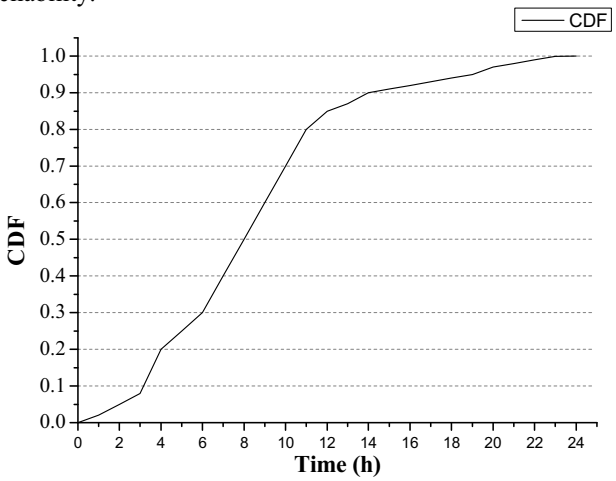


Figure 4. The cumulative distribution function of data access

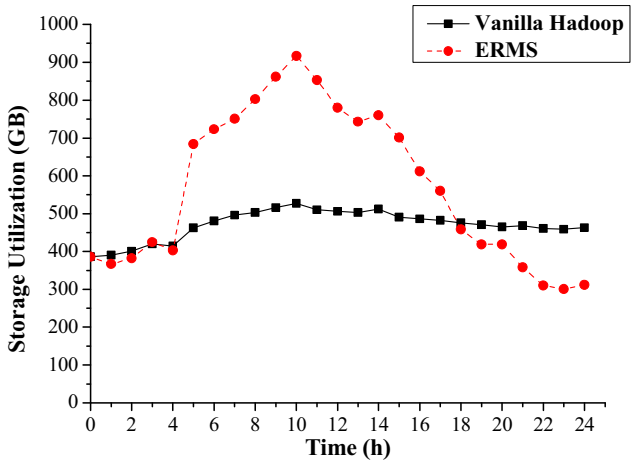


Figure 5. Storage Space Utilization

C. *System Metrics*

Replication factor is a key issue of replication management in HDFS. It could obviously affect system performance. TestDFSIO evaluates the I/O performance of HDFS. It does this by using a MapReduce job as a convenient way to read or write files in parallel. Each file is read in a separate map task, and the output of the map is used for collecting statistics relating to the file just processed. The statistics are accumulated in the reduce task, to produce a summary.

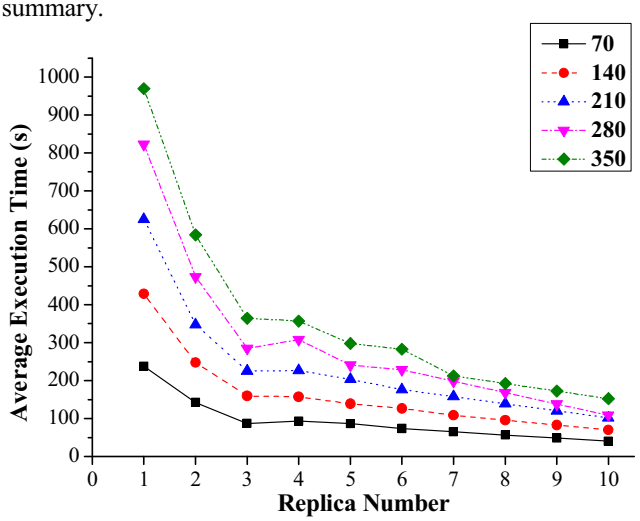


Figure 6. The Performance of TestDFSIO

We evaluated TestDFSIO Reading under different replication factor, as shown in Fig. 6. We used different number of concurrent threads (from 70 to 350) to read the same data, and examined the average execution time of these jobs. The results show that high concurrent reading threads decrease the system performance, while high replication factor could increase system performance. The elastic replication management for HDFS is significantly valuable.

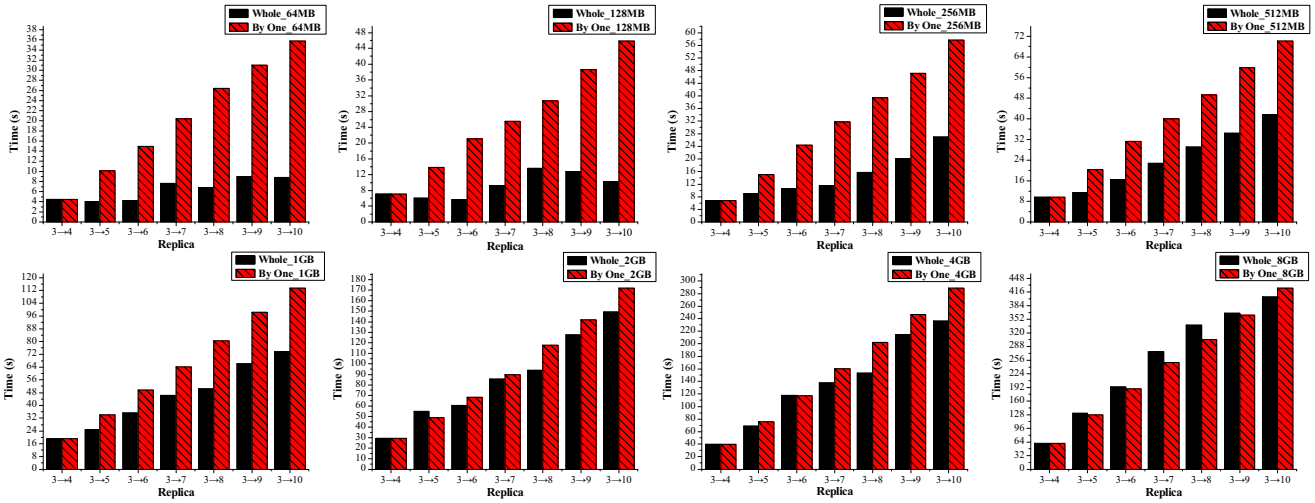


Figure 7. Increasing Replica

There are two ways to increase replicas: increasing the replica directly to the optimal one or increasing replica one by one. Fig. 7 shows the experiment of these two cases under different file sizes. It is clear that increasing the replica directly to the optimal one is a better choice. ERMS figures out optimal replica for hot data, and then increase the extra replicas directly.

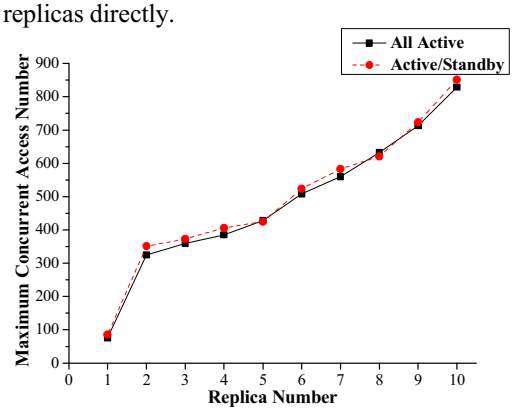


Figure 8. The Maximum Concurrent Access Number the Relicas Could Hold (The File Size is 1GB)

The performances of Map/Reduce applications heavily depend on schedule policy and cluster configuration. To eliminate these effects, we directly read data from HDFS instead of by Map/Reduce framework.

We examined the maximum concurrent access number the replicas could hold in two situations. The first is to keep all eighteen datanodes active, and the other is to keep ten datanodes active and eight datanodes standby. The results are shown in Fig. 8. The maximum concurrent access number of each replica could hold is 80-100, so the maximum of τ_M in our environment 80.

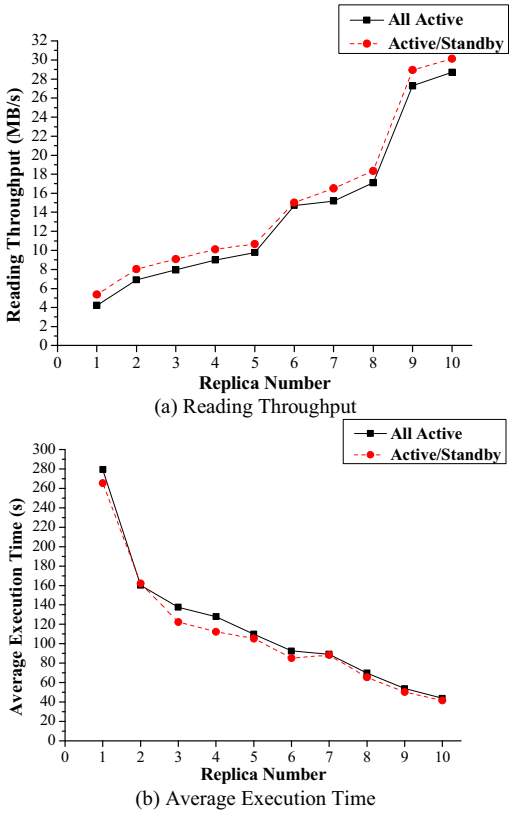


Figure 9. Reading Throughput and Average Execution Time of Reading Benchmark (The File Size is 1GB)

We also evaluated the reading throughput and average execution time when the concurrent access requests are 70. Fig. 9(a) and Fig. 9(b) show that the Active/Standby Model is better than keeping all nodes active. It also indicates that high replication factor could also increase system performance in these situations.

V. Conclusion and Future Work

Data replication is a technique commonly used to improve data availability and reading throughput in distributed file systems. Statistical results show that data access patterns in HDFS clusters are heavy-tailed. Some data are considerably more popular than others, while some are cold. The current replication mechanisms that replicate data a fixed number are inadequate for the varying data access patterns.

In this paper, we present the design and implement of ERMS, an elastic replica management system for HDFS that seeks to increase data locality by replicating the hot data while keeping a minimum number of replicas for the cold data. ERMS dynamically adapt to changes in data access patterns and data popularity, and impose a low network overhead. The active/standby storage model and replica placement strategy used by ERMS would enhance the reliability and availability of data.

In the future, we plan to investigate more effective solutions to detect and predict the real-time data types. We also prepare to evaluate ERMS in real cloud systems, which are provide by Tencent and HuaWei.

Acknowledgment

This work was partially supported by the National High Technology Research and Development Program ("863"Program) of China under the grant No.2011AA01A203, International Science & Technology Cooperation Program of MOST under the grant No.2009DFA12110, National Science Foundation of China under the grant No. 61133004.

References

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST 10), 2010, pp. 1-10.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," in Proceedings of the 6th conference on Symposium on Opearting Systems Design Implementation, 2004.

[3] S. Cheng, Z. Cheng, Z. Luan, and D. Qian, "NEPnet: A scalable monitoring system for anomaly detection of network service," in Proceedings of the 7th International Conference on Network and Service Management (CNSM 11), 2011, pp. 1-5.

[4] Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience: Research Articles," Concurrency and Computation: Practice and Experience, vol. 17, pp. 323-356, 2005.

[5] HDFS Architecture Guide http://hadoop.apache.org/common/docs/stable/hdfs_design.html

[6] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "CDRM: A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster," in Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER 10), 2010, pp. 188-196.

[7] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: coping with skewed content popularity in mapreduce clusters," presented at the Proceedings of the sixth conference on Computer systems, EuroSys '11 Salzburg, Austria, 2011.

[8] L. Abad, Y. Lu, and R. H. Campbell, "DARE: Adaptive Data Replication for Efficient Cluster Scheduling," in Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER 11), 2011, pp. 159-168.

[9] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "DiskReduce: RAID for data-intensive scalable computing," in Proceedings of the 4th Annual Workshop on Petascale Data Storage, Portland, Oregon, 2009.

[10] 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." Conference on File and Storage Technologies (FAST), USENIX, 2012.

[11] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer, "Apache hadoop goes realtime at Facebook," in Proceedings of the 2011 international conference on Management of data, Athens, Greece, 2011.

[12] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in Proceedings of the 7th international conference on Autonomic computing, Washington, DC, USA, 2010.

[13] F. Leibert, J. Mannix, J. Lin, and B. Hamadani, "Automatic management of partitioned, replicated search services," in Proceedings of the 2nd ACM Symposium on Cloud Computing, Cascais, Portugal, 2011.

[14] 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 Proceedings of the 9th USENIX conference on Operating systems design and implementation, Vancouver, BC, Canada, 2010.

[15] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, and H. Wang, "Benchmarking cloud-based data management systems," in Proceedings of the 2nd international workshop on Cloud data management, Toronto, ON, Canada, 2010.

[16] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in Proceedings of the 7th symposium on Operating systems design and implementation, Seattle, Washington, 2006.

[17] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance Using Workload Suites," in Proceedings of the 2011 IEEE 19th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2011.

[18] Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," SIGOPS Oper. Syst. Rev., vol. 44, pp. 35-40, 2010.

[19] M. Y. Eltabakh, Y. Tian, F. Ozcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: flexible data placement and its exploitation in Hadoop," Proc. VLDB Endow., vol. 4, pp. 575-585, 2011.

[20] Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. S. Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in Proceedings of the 2010 international conference on Management of data, Indianapolis, Indiana, USA, 2010.

[21] M. K. McKusick and S. Quinlan, "GFS: Evolution on Fast-forward," Queue, vol. 7, pp. 10-20, 2009.

[22] S. Ghemawat, H. Gobioff, and S. T. Leung, “The Google file system,” in Proceedings of the 9th ACM Symposium on Operating Systems Principles (SOSP 03), Bolton Landing, NY, USA, 2003.

[23] J. Shafer, S. Rixner, and A. L. Cox, “The Hadoop distributed filesystem: Balancing portability and performance,” in Proceedings of the IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS 10), 2010, pp. 122-133.

[24] E. Wu, Y. Diao, and S. Rizvi, “High-performance complex event processing over streams," in Proceedings of the 2006 ACM SIGMOD international conference on Management of data, Chicago, IL, USA, 2006.

[25] W. Tantisiriroj, S. W. Son, S. Patil, S. J. Lang, G. Gibson, and R. B. Ross, “On the duality of data-intensive file system design: reconciling HDFS and PVFS,” in Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, Washington, 2011.

[26] Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas, “Windows Azure Storage: a highly available cloud storage service with strong consistency,” in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, Cascais, Portugal, 2011.