A Novel Approach to Improve Availability of Massive Database Systems (MDS)

Zhengbing Hu
Huazhong Normal University,Department of Information Technology,Wuhan,China
Email:kievpastor@yahoo.com

Kai Du
National University of Defense Technology, Changsha, China
Email: keyes.du@gmail.com

***Abstract*—Because of the huge scale and numerous components, a massive database system's availability has become a serious challenge. Many database replication technologies are used to increase the MTTF, but few are provided to decrease MTTR in massive database systems where the traditional backup methods are not feasible for expensive human cost. Based on analyzing the characteristics of the data in massive databases, we propose a novel approach called Detaching Read-Only (DRO) mechanism and its variation DRO+. It decreases MTTR through reducing the size of physically changing data in every database by detaching data on node granularity. The analysis and experiment results show that our approach can not only reduce MTTR by an order of magnitude, but also reduce the expensive human cost without extra hardware cost.**

***Index Terms*—Database, Massive Database Systems, MTTR**

I. INTRODUCTION

Your goal is to simulate the usual appearance of papers in a Journal of the Academy Publisher. We are requesting that you follow these guidelines as closely as possible. The requirements to store and query massive data in scientific and commercial applications have appeared. Alexander Szalay and Jim Gray address that the amount of scientific data is doubling every year and scientific methods are evolving from paper notebooks to huge online databases [1]. Until 2000, disk capacity has improved 1,000 fold in the last 15 years, consistent with Moore's law, so storage volume is not the primary challenge as before [2]. System maintenance, automation and availability have become the new great challenges [3].

It is a good idea to build a massive database system with federated databases [4]. Because of the complexity of management and maintenance of a single PB(PetaByte)-scale massive database system, partitioning it into many small federated databases is a feasible way. However maintaining such a huge system is expensive due to its low availability caused by its large scale.

There are at least two challenges in gaining the high availability in massive database systems: the short MTTF (Mean Time To Failure) and long MTTR (Mean Time To Recovery) [5] for storage failures. The former is caused by the number of hardware components. For example, if the availability of a database of 1TB is 99.99%, the availability of a database system of 100TB constituted of 100 such databases will only be 99%. The latter is caused by single database's size. There are two reasons: 1) Recovering the data of 1TB needs a long time even if with fine backup solutions like archived and timely backup [6]. 2) Finely backuping 100 databases is a huge DBAs' cost.

The efficient way to increase the overall MTTF is to increase the MTTF of single database in a massive federated system since the number of databases can't be decreased. How to increase MTTF has been researched by those who mainly focus on how to provide efficient synchronization mechanism between the replicas of a cluster [8] and care little about the recovery time.

Decreasing the MTTR for storage failure is a great challenge for the database's size. Little attention has been paid to this problem in previous research because it is a new problem which only comes forth in a massive system. The idea of reducing MTTR rather than increasing MTTF has been proposed in the ROC [9] project. How to decrease the MTTR and human cost will be researched in this paper.

From the analysis above, we will achieve two objectives in a massive system:

- To improve the massive database system's availability by reducing the MTTR.
- To decrease the MTTR without extra expensive human cost.

Currently the massive storage system is usually filled by the high-rate streaming log data, such as science experiment data [10,11], call detail records, RFID-based free way tracking [12], network packet traces. All these data basically have the same features:

- The data are generated high-rate and continuously (otherwise the information will not be cumulated into a massive system of TB or PB-scale).
- The data are just appended into the system and the old data will not be updated.

We call these features as "insert-once-no-update". Leveraging these features, we design a novel mechanism DRO and its variation DRO+ to separate the insert-once-

doi:10.4304/jsw.4.10.1145-1151

no-update "read-only" data from the online loading data. We compare the novel mechanism to the double replication used in an existed massive system in MTTR, TCO (Total Cost of Ownership) and performance. The results show that DRO+ excels others in most cases.

The organization of this paper is as follows: Section II describes a massive database system CDRMDB. Section III describes our novel replication mechanisms. Section IV and Section V analyze the MTTR and cost of the novel mechanism; Section VI is the experiment; Section VII is the related work and Section VIII is the conclusion and future work.

II. A MASSIVE DATABASE SYSTEM: CDRMDB

In this section, we will illustrate a massive database system CDRMDB (Call Detail Records Massive DataBase) which stores high-rate and massive call detail records. It is built up with 86 database nodes and every node's volume is 500GB. CDRMDB has the following features which are classic to many massive streaming data storage systems: 1) Store high-rate and massive streaming data. 2) Provide query access interface to the massive data. 3) The scalability and availability are two key system features.

In CDRMDB, the simple Primary Backup mechanism [13] is adopted in terms of efficient loading performance. One primary and backup replica constitute one cluster. So it is constituted by 43 database clusters.

The cause of too long MTTR is database size and recovery mechanism. In CDRMDB, one database' size is 500GB. During the recovery period, all the 500GB data must be loaded into the recovered database from the correct database through logical export and import mechanism. The size and logical recovery mechanism lead to the long MTTR. In addition, double replication may cause the failure cluster to be the bottleneck of the query process when the recovery is being done because the exporting data operation will dramatically decrease the system's performance.

In order to eliminate the query bottleneck in Double Replication, Treble Replication is a better choice with more half cost of the former. When a cluster has three replicas and one of them fails, one normal database is used to recover the data and the other can process the query as before. Another benefit of the treble replication is that the cluster's availability is higher because the probability of three databases failing is lower by an order of magnitude than two.

III. NOVEL REPLICATION MECHANISMS: DRO & DRO+

In this section, firstly we uncover the inherent reasons of too long recovery time in massive database systems in Section III.A. Then we propose a novel replication mechanism DRO and its variation DRO+ in Section III.B and III.C.

A. Why So Long Recovery Time

In Section II we show the conflict of decreasing the MTTR and decreasing the maintenance cost. The essential reason of the conflict is that the system's scale is so large that the traditional backup can't be done because the DBAs' cost is high.

The main idea of shortening the recovery time stems from the feature of insert-once-no-update which is described in section I. We can explain this idea clearly in Fig.1. In Fig.1 (a) and (a1), it shows the current state of CDRMDB --all data is in an online changing database. The databases' data is always changing and all loading and query requests are issued to the total database. If a database fails for storage failures, it needs to recover all data whose size can reach 500GB in CDRMDB. It will take several days.

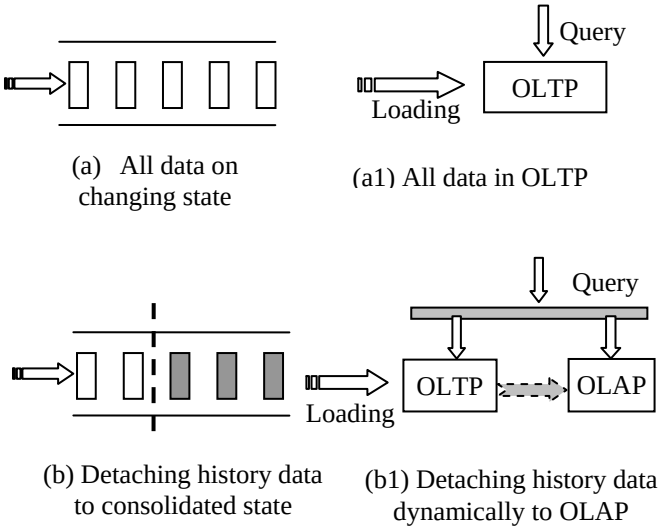

Figure 1. Online detaching history data

However this can be avoided by leveraging the insert-once-no-update feature. We can periodically detach the history data which will not be updated. This idea is shown in Fig.1 (b) and (b1). Fig.1 (b) means to logically detach the history data from a database. Fig.1 (b1) means to divide the system into two parts from the system's view. One part processes loading and query requests as an OLTP database. The other stores history data and it is consolidated as an OLAP database. In addition, in order to eliminate the IO contention between detaching and loading, it is sound to detach data based on node.

B. Detaching Read-Only Data Replication

To detach history data based on node granularity can be implemented by dynamically deploy nodes to different usages. We call this as "Detaching Read-Only (DRO) Replication". The idea stems from three basic facts: 1) a read-only database can have an extremely short recovery time comparing to a changing database. 2) The smaller the scale of a write-read database, the shorter the recovery time.3) some research about read-only compressed database [14] has proved that compressed databases will likely do better.

1. Tasks in DRO

Now let's illustrate the DRO mechanism in Fig. 2. In Fig.2, the system's running time is divided by cycles. In every cycle, two works will be done parallelly: loading

data into the write-read database clusters and compressing read-only database clusters. In cycle 1, three database clusters which are called as *Loading Database Cluster (LDC)* (marked as "**Loading**") provide query and data loading functions. In cycle 2, the three database clusters become read-only, and at the same time another three clusters are added into the system as LDCs. The three read-only database clusters are compressed and all the data is collected into one database cluster which is called *Query Database Cluster (QDC)* marked as "**Compressed & Query**" with black color. The two free database clusters which are released after data compressing (boxes with dashed frame in Fig.2) will be added into the third cycle for next cycle's data loading task. In cycle 3, one fresh database cluster which is newly added into the system (boxes with solid frame in Fig.2) and two free database clusters from cycle 2 make up the new data LDCs. Just like in cycle 2, two free database clusters come out after compressed and are added into cycle 4. In cycle 4, the procedure is continued and the rest may be deduced by analogy.

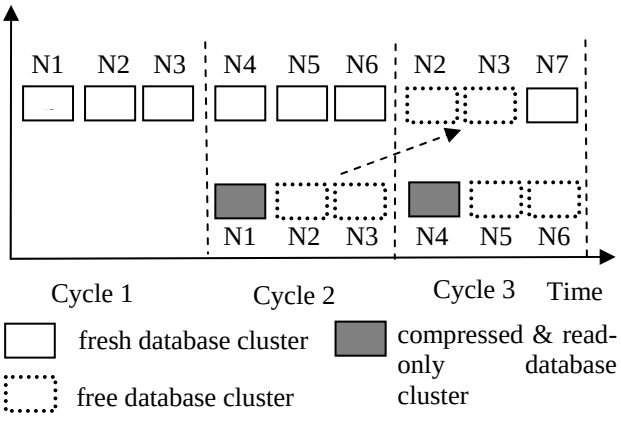

Figure 2. Online detaching history data

2. Two Types of Database Clusters

In a *LDC*, all the database nodes save the same data but the data are not stored in the absolutely same physical files. For example, when a tuple R is loaded into the three databases d1, d2, d3 in a cluster, it may be saved in file1 in d1, file2 in d2, file3 in d3 while it is saved logically in the completely same way. So when one database node breaks down for a media failure, the database needs to be built from the blank and imports the data which are exported from another normal node.

However, in a *QDC*, all database nodes save absolutely the same physical data. When a compressed database node has been created, it can be copied into another node through a disk copy: copying all the files with the same directories and files. This mechanism profits from the data's no-update-after-insert property. Its advantage is that when a database node breaks down for media failures, only the fault files need to be copied from the normal node. It shortens the recovery time and tinily affects the normal node.

C. DRO+: improved DRO

DRO has decreased the MTTR without increasing DBAs' backup and recovery work. However it loses much loading performance because too few nodes are used to load data in every cycle. In DRO, all database nodes have the same storage volume and the storage resources are wasted since its volume is designed for all online time which is larger than one cycle. So we can save the budget for storage to buy more *Loading Database nodes.* The procedure of DRO+ in Fig. 4 is like DRO except that one compressed and read-only database cluster is added in every cycle whose storage size is larger than *LDCs* and in every cycle the number of *LDCs* is equal to the treble replication. What is the economical benefit of DRO+ over DRO will be illustrated in Section IV.

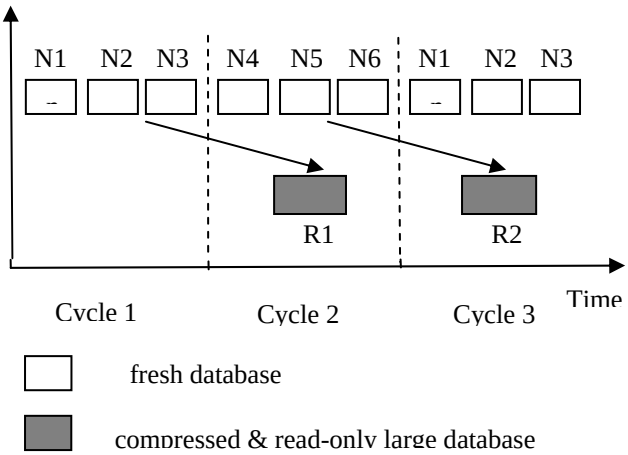

Figure 3. Online detaching history data

IV. MTTR ANALYSIS

In Section III, we have illustrated four database replication mechanisms. In this section, we will analyze MTTR, performance, economic cost of the four mechanisms.

A. MTTR in DRO

In order to analyze the system's MTTR in DRO, we define the following variables: The number of database clusters is N_S. The data's online time is T. The time of every cycle is C. The number of *LDC* in every cycle is N_L. The storage volume of one *LDC* is S. The requested average data loading rate is V. The overall performance of the data loading is P_L. The compression ratio is R_1. The space utilization ratio of every database node is R_2 after it becomes a query-only database and before it is compressed.

In Table 1, the number of various types of database clusters in every cycle is shown. The value of every column in Table 1 is calculated based on the volume usage. The number of fresh database clusters should be an integer which is not less than the float value $N_L * R_1 * R_2$, and the number of free database clusters should be an integer which is not greater than $N_L * (1 - R_1 * R_2)$.

For example, in Fig.2, the value of every variable is like the following: $N_L = 3$, $R_1 * R_2 = 1/3$, $N_L * R_1 * R_2 = 1$, $N_L * (1 - R_1 * R_2) = 2$.

TABLE I.
NUMBER OF VARIOUS TYPES OF DATABASE CLUSTERS IN EVERY CYCLE

Cycle No	# of fresh database clusters	# of free database clusters
1	N_L	0
2	N_L	$N_L * (1 - R_1 * R_2)$
3	$N_L * R_1 * R_2$	$N_L * (1 - R_1 * R_2)$
4	$N_L * R_1 * R_2$	$N_L * (1 - R_1 * R_2)$
…	$N_L * R_1 * R_2$	$N_L * (1 - R_1 * R_2)$

In every cycle one or more QDC is created after the database clusters in the previous cycle are compressed. Because a QDC can only be written once with the compressed data and will not allow to be updated, the following two points should be guaranteed:

1) To make the best of the storage space, a QDC should be filled up as much as possible since the QDC can only be written once.
2) The cycle should be long enough that the volume of the data which is loaded in this cycle in the N_L LDCs is large enough to fill up the storage space of one or more QDCs after the data is compressed.

According to the above analysis and Table 2, we can educe the following equations:

$$N_S = N_L + N_L + (T/C - 2)*N_L*R_1*R_2 = (2 + (T/C - 2)*R_1*R_2)\,N_L. \quad (1)$$

The data loading performance is generally thought as relative to the number of the data loading clusters, so we have this: (f_1, f_2 is a constant factor.)

$$P_L = f_1 * N_L. \quad (2) \qquad V = f_2 * P_L. \quad (3)$$

The system's MTTR is classified as two types: $MTTR_Q$, the MTTR of one QDC; $MTTR_L$, the MTTR of one LDC. Since the recovery operation of a QDC is only to copy one or more data file, the recovery time is a constant time t_0. So the system's MTTR is determined by the $MTTR_L$. In order to simplify the discussion about the $MTTR_L$, we assume that it is linear to the data size of one node of the LDC: (f_3 is a constant factor.)

$$MTTR_L = f_3 * V * C / N_L.$$

Referring to (2), (3), we can get:

$$MTTR_L = f_1 * f_2 * f_3 * N_L * C / N_L = f_1 * f_2 * f_3 * C. \quad (4)$$

From (1), (4), we can find the relation between $MTTR_L$, C and N_L:

$$MTTR_L = f_1 * f_2 * f_3 * T * R_1 * R_2 / (2 * R_1 * R_2 - 2 + N_S / N_L). \quad (5)$$

Theorem 1. In DRO and DRO+, assuming that the data loading performance is proportional to the number of data loading clusters and the MTTR of one database is proportional to its size, the MTTR is proportional to the cycle C.

Proof. As discussed above. □

Theorem 1 indicates that C determines the MTTR.

B. MTTR and Performance Comparison

Theorem 2. In Treble, DRO and DRO+, assuming that the data loading performance is proportional to the number of LDCs and the MTTR of one database is proportional to its size, the MTTR of DRO+ is smallest and the loading performance of DRO+ is best.

Proof. We mark the MTTR of treble, DRO and DRO+ as $MTTR_T$, $MTTR_D$ and $MTTR_+$. In fact, the double and treble replication mechanisms are a special case of DRO from (4). In (4), the treble replication means $R_1 = R_2 = 1$, $T = C$, so the $N_S = N_L$, $MTTR_T = f_1 * f_2 * f_3 * C = f_1 * f_2 * f_3 * T$. From (4), we also get that $MTTR_D = MTTR_+ = f_1 * f_2 * f_3 * C$. So $MTTR_D : MTTR_+ : MTTR_T = C : C : T$ (6). If we assume the loading performance is linear to the number of *Loading Clusters*, obviously we can get the comparison of Performance of Treble, DRO and DRO+: $P_T : P_D : P_+ = N_S : N_L : N_S$. (7) From (6) and (7), we can infer that DRO+ has the smallest MTTR and the best performance.

V. MTTR ANALYSIS

In Section IV.B we have concluded that DRO+ has the smallest MTTR and the best performance. The cost may be higher than the other two. In this section we will show that the total cost of DRO+ is not always more than the other two.

A. Original Cost Analysis

The original hardware cost includes CPU, memory, storage, network switch, and so on. In order to simplify analyzing the cost of the three replication mechanisms, we assume that every node has the same number of CPU and memory and the storage's cost is proportional to its volume.

Theorem 3. If the ratio of the storage cost to the computation cost is more than some value, the original cost of DRO+ is not more than Treble and DRO.

Proof. The original hardware cost of Treble, DRO and DRO+, C_T, C_D, C_+ is:

$$C_T = C_D = 3 * N_S * (CPU + S) \quad (8)$$

$$C_+ = 3 * (2 * N_S * (CPU + S_W) + (T/C - 1) * (CPU + S_R)) \quad (9)$$

CPU is one node's cost except its storage cost. The other variables are defined in Section IV.A. In (9), one loading node's storage size is $S_W = (C/T) * S$, one read-only node's storage size is $S_R = N_S * S_W * R_1$. So if we expect the cost of DRO+ is not more than the other two, that is $C_+ <= C_T = C_D$, we should keep the following inequality:

$$S/CPU >= (N_S + T/C - 1))/(N_S * (1 - R_1 + (C/T) * (R_1 - 2))) \quad (10)$$

From (10), we can conclude that if the ratio of storage to CPU cost is larger than $f(C) = (N_S + T/C - 1))/(N_S * (1 - R_1 + (C/T) * (R_1 - 2)))$, the original cost of DRO+ will not overspend the other two. □

Through simple analysis of f(C), we can find that it has a min value when $R_1 > 0.1$ and $T/C > 3$ are true which is really true in most cases. In Section VI.B we can find it really true.

B. TCO Analysis

The total cost of ownership (TCO) [15] of an information system can be divided into two parts: original hardware cost and management cost. The former is discussed in Section V.A and the latter mainly is human cost. As G. Weikum points out, TOC in a mission-critical system becomes more and more dominated by the money spent on human staff [16].

The human cost in recovery can be calculated as the total recovery time: failure count*MTTR. Since the system storage scale is the same, the failure count is the same. So the ratio of the human cost of DRO+ to Treble is $CH_+:CH_T = MTTR_+: MTTR_T = C:T$. The cost of transforming online data to offline is difficult to analyze in quantity, but at least it is clear that it is zero cost in DRO+ or DRO discussed in Section III.C.2. In Treble, it is expensive for it needs huge extra human operation such as exporting and deleting old online data and it will sharply degrade the performance of a 24*7 running system.

VI. Simulation and Cases Analysis

In this section, we will show the differences of MTTR and cost in the four mechanisms which are described in section V through simulating several massive systems.

A. MTTR and Performance

In this section we pay attention to MTTR and NL. Fig.4 shows the value of NL with different cycle. NS = 30, T = 150 days, R1 =0.4, R2 =0.5, f1= f2 =1. For treble and double replication, C=T=150 days; for DRO, C=10,15,30,50 days, NL is calculated from (1). In Fig.5, MTTR is calculated from (4) in Section IV.A. For Treble and DRO, f3 = 0.1 and for Double, f3 = 0.13. For Double and Treble, C=T=150 days. For DRO, MTTR varies with the cycle. From Fig.4 and 5, we can conclude that the shorter the cycle is, the shorter the MTTR is, but the data-loading performance the worse may be. MTTR can be decreased to one-tenth when the cycle is 15 days.

B. Cost Analysis

Fig.6 is the curves of the right expression of (10). Fig.7 is the human cost ratio of DRO+ to Treble in recovery calculated from Section V.B. In Fig.6, T=150 days, the curves show that the larger N_S is and the smaller R_1 is, the smaller the ratio is. That is to say the larger the system is and the higher the compression ratio is, the more possibly the hardware cost in DRO+ is not more than the others.

When C=15, N_S =30 and R_1=0.1, the minimum ratio reaches 1.8 and the human cost in recovery is only one-tenth in DRO+. The value of 1.8 can be easily reached in data-intensive applications like TPCC. The NO.1 in TPCC's Price/Performance column until 27-May-2006 is Dell's PowerEdge 2800 whose S/CPU is 1.95 [17]. This is evidence that the original cost of DRO+ is quite probable to be not more than Treble or DRO.

VII. Related Work

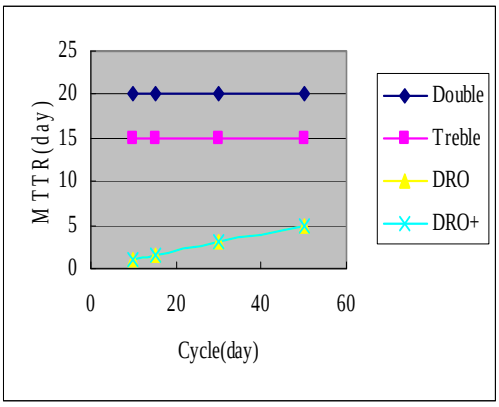

Figure 4. Performance and Cycle

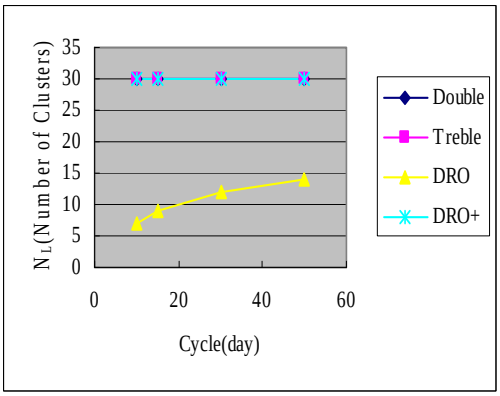

Figure 5. MTTR and Cycle.

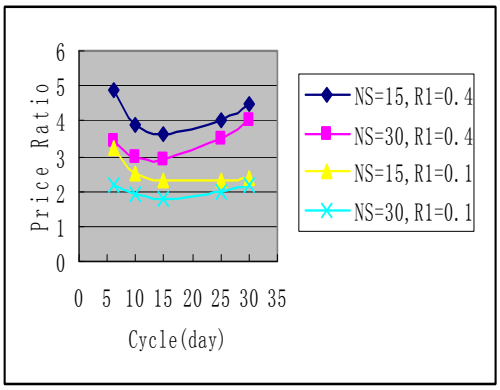

Figure 6. Ratio of S/CPU in DRO+

In the research community of database availability, database replication is a primary technology. The eager approach may easily lead to deadlocks and be hard to scale for its block mode [18]. So many lazy approaches are proposed to improve the overall performance and scalability [19]. However, all of them discuss little about the recovery of a replica from other replicas in a massive scale background.

Another novel research point is to build a high availability system based on share-nothing database clusters. RAIDb [7] aims to build a high available system like the commercial system based on multiple open-source databases. It implements the high availability and

load schedule through building the middleware tier, caches all the sql operations in the management node and redoes them on the fault database replica. This recovery way is not feasible when the data arrives at a high rate because the cache size is too large and the traditional backup mechanism is not useful to each replica.

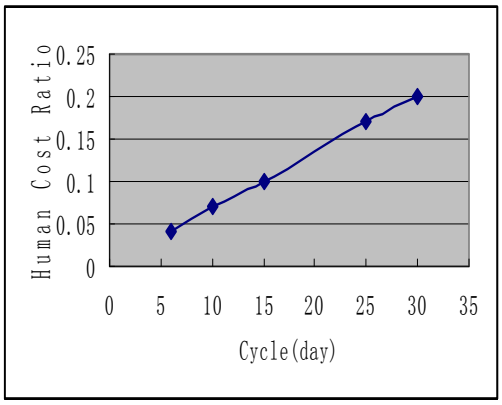

Figure 7. Human Cost Ratio of DRO+ to Treble.

Oracle RAC [20] adopts another way to gain high availability on massive systems. It provides high availability through a multiple instances fault-tolerance mechanism in the query processing tier and can't continue working when encountering media failures. So it can't easily provide high availability for the high media fault ratio in a massive system.

Google is a successful massive system. It is constituted by about ten thousand nodes. About one hundred nodes break down every day [21]. Its data scale is 40-80TB. The method to gain high availability is data replication. All data and metadata is replicated double or treble. It focuses on the high availability of a massive file system and its recovery granularity is physical files. Thus the difficult problem of data consistency in database recovery doesn't exist.

The replication mechanisms proposed in this article focus on quick recovery from the media failure node which is not covered enough in the above research work. Especially the view of taking the total cost of recovery into account is not addressed earlier.

VIII. Conclusions

The development of computing technology in several decades has made it possible to store massive data like web-log, call detail records, sensor data. However, the short MTTF and long MTTR of massive systems caused by the massive scale becomes a new challenge. Much work has been done to increase MTTF but little attention has been paid to decrease MTTR which is a severe problem in running product systems. Based on the experiences of our product system CDRMDB, we propose a novel mechanism DRO and its variation DRO+ from a systemic and economical view. The simulation shows that our approach can sharply decrease the MTTR by an order of magnitude without any performance loss and need no extra hardware or human cost.

Acknowledgment

The authors wish to thank Prof. V.P. Shirochin.This research was supported by China Postdoctoral Science Foundation(20070420908) and by the Project-sponsored by SRF for ROCS, SEM (2008890).

References

[1] Jim Gray, Alex Szalay. Science in an exponential world. Nature,V.440.23, 2006.
[2] Jim Gray, Prashant Shenoy. Rules of Thumb in Data Engineering. ICDE2000.
[3] Jacek Becla, Daniel Wang. Lessons Learned from Managing a Petabyte. CIDR2005.
[4] Boris Gelman. V2LDB. CIDR2005.
[5] K. Nagaraja , X. Li and B. Zhang, R. Bianchini, R. Martin and T. Nguyen. Using Fault Injection and Modeling to Evaluate the Performability of Cluster-Based Services. In Proceedings of the Usenix Symposium on Internet Technologies and Systems, Mar. 2003.
[6] Abraham Silberschatz, Henry F. Korth, S. Sudarshan. Database System Concepts. 4th edition. China Machine Press. P461-470.
[7] Emmanuel Cecchet. C-JDBC: a Middleware Framework for Database Clustering. IEEE Computer Society Technical Committee on Data Engineering 2004.
[8] Yuri Breitbart, Raghavan Komondoor, Rajeev Rastogi, S. Seshadri, Avi Silberschatz.Update Propagation Protocols For Replicated Databases. SIGMOD 1999.
[9] Patterson D. A., A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, N. Treuhaft. Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. UC Berkeley Computer Science Technical Report UCB//CSD-02-1175, March 15, 2002.
[10] Y. Dora Cai, Ruth Aydt, Robert J. Brunner. Optimized Data Loading for a Multi-Terabyte Sky Survey Repository. In Proc.Supuer Computing 2005.
[11] A. Szalay, P. Kunszt, A. Thakar, J. Gray, R. Brunner. Designing and Mining Multi-Terabyte Astronomy Archives: The Sloan Digital Sky Survey. In Proc. SIGMOD2000.
[12] http://www.511.org/fastrak
[13] Matthias Wiesmann, Fernando Pedone, Andr′ e Schiper, Bettina Kemme,Gustavo Alonso. Transaction Replication Techniques: a Three Parameter Classification. SRDS 2000.
[14] Daniel J. Abadi, Samuel R. Madden, and Miguel C. Ferreira. Integrating Compression and Execution in Column-Oriented Database Systems. Proceedings of SIGMOD 2006.
[15] Hitt, Ellis F. Total ownership cost use in management. Digital Avionics Systems Conference 1998.
[16] Gerhard Weikum, Axel Moenkeberg, Christof Hasse, Peter Zabback. Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. VLDB2002.
[17] http://www.tpc.org/tpcc/results/tpcc_result_detail.asp?id=105092601
[18] Jim Gray, Pat Helland, Patrick O'Neil and Dennis Shasha - The Dangers of Replication and a Solution.ACM SIGMOD 1996.
[19] A. Sousa,J. Pereira, L. Soares, A. Correia Jr., L. Rocha, R. Oliveira, F. Moura.Testing the Dependability and Performance of Group Communication Based Database

Replication Protocols. Dependable Systems and Networks (DSN) 2005.
[20] Building Highly Available Database Servers Using Oracle Real Application Clusters. An Oracle White Paper May, 2001.
[21] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. The ACM Symposium on Operating Systems Principles (SOSP) 2003.

Zhengbing Hu was born in 1978. He received B.E., M.E. and P.h.D degree in National Technical University of Ukraine. His current research interests include Network Security, Intrusion Detection System, Artificial Immune System, Data Minging etc..

Kai Du was born in 1978. He received B.E. and M.E. PhD degree in National University of Defense Technology, China. His current research interests include large-scale data management, data reliability, distributed computing.