論文 / 著書情報
Article / Book Information

Title	Performance and Reliability of a Revocation Method Utilizing Encrypted Backup Data
Author	Kazuki Takayama, Haruo Yokota
Journal/Book name	Proc. of IEEE Pacific Rim International Symposium on Dependable Computing 2009, , , pp. 151-158
発行日 / Issue date	2009, 12
権利情報 / Copyright	(c)2009 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

2009 15th IEEE Pacific Rim International Symposium on Dependable Computing

Performance and Reliability of a Revocation Method Utilizing Encrypted Backup Data

Kazuki Takayama
Department of Computer Science
Tokyo Institute of Technology
Tolyo, Japan
Email: takayama@de.cs.titech.ac.jp

Haruo Yokota
Global Scientific Information and Computing Center
Tokyo Institute of Technology
Tokyo, Japan
Email: yokota@cs.titech.ac.jp

***Abstract*—When multiple users access a network storage system for cloud computing, security becomes a key factor in the service, as well as performance and reliability. The "encrypt-on-disk" scheme effectively protects transmitted and stored data in network storage. However, this scheme has the problem of revocation for shared files. Active revocation is safe but has denial periods to allow immediate reencryption, while lazy revocation has no denial period but is unsafe during the delay. We propose intelligent storage nodes capable of handling active revocation in storage without the denial period by adopting a primary–backup configuration. This approach provides a good combination of security and availability by replication. However, the reencryption process negatively affects the update performance. Delaying the reencryption process and disk write on the backup node improves performance with no ill effect on security and a small decrease of MTTDL for the simple primary–backup configuration. We evaluate the performance of the proposed approaches by experiments, and the reliability by estimation.**

***Keywords*-secure storage, revocation, encrypt-on-disk, parallel storage, primary–backup structure**

I. Introduction

Network storage is undoubtedly a key component of cloud computing. The scalability, availability and security of the storage systems are important requirements for the cloud computing [1]. There are many types of network storage systems, including those based on Redundant Array of Independent Disks (RAID). However, the most popular RAID types (types 3–6) have scalability and maintainability problems. The primary–backup configuration guarantees bandwidth and response time even under disk failures, and even in a large system. A combination of chained declustering data placement [2] as a form of primary–backup and a distributed directory is effective to derive scalable bandwidth and low latency.

For the security of the network storage system, data transferred between the network storage system and clients, or within the network storage system itself, must be protected from malicious attacks. To implement secure storage systems, several data protection functions have been proposed [3]. The so-called "encrypt-on-disk" scheme protects the transferred data by encrypting the data as they are stored to disk, while the "encrypt-on-wire" scheme performs the encryption on the fly as the client requests the data. The former scheme is safer and provides better performance than the latter, because it does not use encryption processes for every read or write operation on the storage side, and it keeps the data in cipher [4]. However, if files in the network storage are shared by multiple users, the encrypt-on-disk scheme must be able to handle the revocation of users. These files should be reencrypted with a new key whenever a revocation occurs. Active revocation is safe but has access denial periods because of the immediate reencryption, while lazy revocation has no denial period but is unsafe during the delay [4], [3].

We have proposed a method of handling active revocation in intelligent storage nodes without the denial period by adopting a primary–backup configuration [5]. Preencrypting backup data with a key different from that used on the primary store and switching the role of primary and backup after a revocation realizes a nonstop safe service. It provides a good combination of security and availability because of the replication. However, it may cause update performance degradation because the encryption taking place on the backup node constitutes an additional load.

Here, we consider delaying the write and reencryption processes on the backup node to improve update performance. In the delayed write approach, the differences caused by recent writes are not written to the backup disk immediately but are kept in memory in the backup node. In the delayed reencryption and write approach, the encryption operations for the updated data are also delayed. These approaches have the benefit of utilizing system resources effectively and aggregating multiple updates for a file but also have the ill effect of decreasing system reliability by keeping data in volatile memory. However, the practical decrease in reliability is very small because of the same effect of the nWAL protocol [6]. To demonstrate this, we evaluate the performance of the proposed approaches by experiments, and the reliability of the system by estimation.

The remainder of this paper is organized as follows. Section II discusses requirements for network storage systems

978-0-7695-3849-5/09 $26.00 © 2009 IEEE
DOI 10.1109/PRDC.2009.32

possessing high performance, availability and maintainability. Section III surveys the current encryption and revocation techniques in network storage. The proposed approaches are explained in Section IV. Section V reports experiments to evaluate performance of the proposed approaches. Section VI estimates their reliability. Finally, Section VII concludes the paper.

II. Requirements for Network Storage Systems

Performance and availability are the most essential properties required for network storage systems for cloud computing. Scalability is also quite significant. A very-large-scale system configuration should have adequate performance for its size while ensuring high availability. The costs of maintaining a large storage system to satisfy performance and availability requirements become a crucial problem. The total amount of storage exceeds the management capability of a human administrator who maintains the system.

RAID 3–6 configurations ensure high availability by adopting parity calculation techniques, but their performance is considerably decreased by the parity calculation for reconstruction if a disk in a parity group fails. Moreover, the scalability of a RAID parity group is restricted for reliability reasons. On the other hand, multiple RAID groups have the problems of load balancing and maintainability.

To realize scalable storage systems possessing high performance, high availability and high maintainability, autonomous disks [7] have been proposed. An autonomous disk system consists of a cluster of highly functional intelligent storage nodes [1] in a networked environment. The intelligent controller in each storage node is responsible for balancing the access load in the cluster and handling failures by migrating data between nodes autonomously. The absence of a centralized controller in the cluster allows high scalability. To migrate data transparently to the clients, a distributed directory is necessary. A fat-Btree [8] is adopted in the autonomous disk system as the distributed directory to provide high accessibility for data stored in all nodes with low update overhead, even in large configurations. This enables each node to migrate data to a neighboring node without extensive modifications to the whole directory structure, and hides changes of data location from the clients.

To make the cluster reliable, the autonomous disk system adopts chained declustering data placement [2] for the primary–backup configuration, where a neighboring node of a primary data node keeps its backup data. Chained declustering is suited to the Btree-based directory structure while allowing data to migrate between the neighboring nodes. The neighboring write-ahead log (nWAL) protocol [6] is effective for asynchronous update of the backup.

[1]The concept of the autonomous disk can easily be applied to a multiple RAID configuration by placing the intelligent controller within each RAID group.

Thus, the approach of the intelligent storage node like the autonomous disk is effective to realize the scalable storage system. To preserve security as well as performance and availability, we consider encryption methods suitable for network storage nodes utilizing the processing facilities on them.

III. Encryption and Revocation in Network Storage

A. Encrypt-on-disk vs. Encrypt-on-wire

There are two encryption schemes to protect data transmitted to or from network storage: encrypt-on-wire and encrypt-on-disk [3]. In the *encrypt-on-wire scheme*, files are stored in clear text and encrypted only when transmitted, using a protocol such as secure socket layer (SSL), while in the *encrypt-on-disk scheme*, files are stored in cipher and transmitted without any further encryption process.

From the viewpoint of data transmission performance, the encrypt-on-disk scheme is more efficient than the encrypt-on-wire scheme, because the storage server in the encrypt-on-disk scheme does not require as much encryption work as the encrypt-on-wire scheme. Moreover, from the security point of view, the encrypt-on-disk scheme protects data while in storage as well as during transmission, while the encrypt-on-write scheme only protects data during transmission.

On the other hand, in a system adopting the encrypt-on-disk scheme for shared files, the files must be reencrypted with a new key when a revocation occurs. This is because revoked users can retain the current key, leading to information leakage if they intercept transmitted files despite their access being denied by access control methods. Therefore, the encrypt-on-disk scheme costs more than the encrypt-on-wire scheme when a revocation occurs.

B. Active vs. Lazy Revocation

Reencryption methods for revocation in the encrypt-on-disk schema are divided into active revocation and lazy revocation according to the timing of the reencryption [4], [3]. In *active revocation*, related files are immediately reencrypted using a new key after the revocation, while the reencryption is delayed until the files are next updated in *lazy revocation*.

Active revocation is more secure than lazy revocation, because revoked users are immediately unable to decrypt the files. However, a weak point of active revocation is that even authorized users cannot access the files until the immediate reencryption processes are completed. This weak point is more pronounced when multiple revocations occur at the same time for different files.

On the other hand, in lazy revocation, because update processes involve encryption, they can be combined with the reencryption required for revocation. In addition, the reencryption work for several revocations may be performed together if the file is not frequently updated. Therefore, lazy revocation is more efficient than active revocation. However,

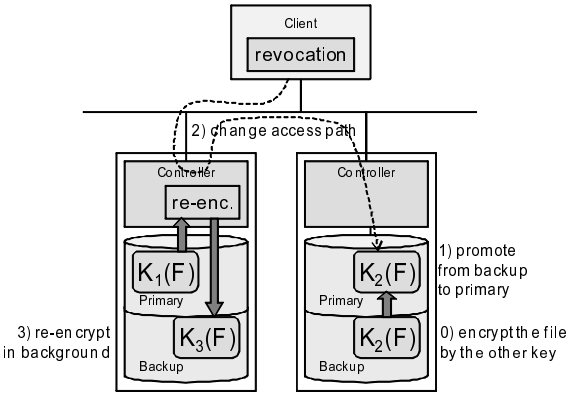

Figure 1. Backup-assisted revocation

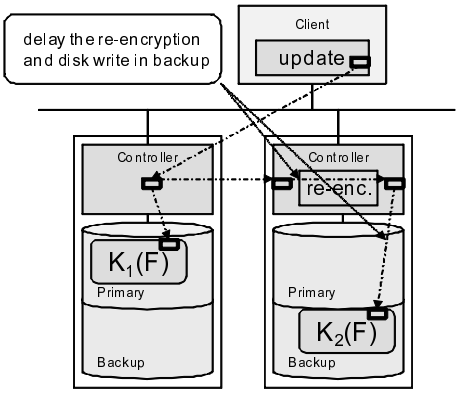

Figure 2. Delayed reencryption and write

lazy revocation introduces a security vulnerability, because files in storage after the revocation and before the update are encrypted with the old key, which may be kept by the revoked users.

IV. Backup-assisted Revocation

A. Outline of the Proposed Approach

To solve the trade-off problem between reencryption methods, we have proposed a method named *backup-assisted revocation* or *BA-Rev*, in which the files to be used after revocation are the backup copies, which have been encrypted with a separate key. Figure 1 illustrates the BA-Rev method, where $K_1(F)$ and $K_2(F)$ mean that the file copies are encrypted using different keys and are treated as the primary and backup, respectively. After a revocation, $K_2(F)$ is promoted to be the primary, and $K_1(F)$ is reencrypted by the new key K_3. If we adopt the chained declustering placement, these files are migrated asynchronously to satisfy any location restrictions. However, it is easy to enable access during the migration by keeping the source during the process.

When a reencrypted file is prepared using a new key, an authorized user given the new key can access the file immediately after the revocation, but unauthorized users cannot. The method is therefore as safe as active revocation because it has no unsafe period, and it provides access for authorized users with no denial period following a revocation.

However, the naive implementation of the BA-Rev is disadvantaged in its update performance, because a reencryption of the backup data is required for every update. The reencryption overhead for updates can be reduced by partial update with block encryption, which does not require reencryption of the whole file. Moreover, the encryption process and disk I/O can be delayed to improve update performance. We call them *delayed write* or *DW* and *delayed reencryption and write* or *DRW*, respectively. When the node storing the primary file receives an update request with a differential portion of the file encrypted by K_1, it updates the primary file and sends the portion to the node storing its backup. The backup node keeps the portion in memory but does not reencrypt it using key K_2, nor does it write it to disk immediately, as shown in Figure 2.

Here, we assume that revocations occur less frequently than it takes to apply the delayed updates and to reencrypt files. It seems reasonable in many cases because the revocation occasionally occurs to change the role of members for some events. Even if revocations occur more frequently, users wait for the end of reencryption just same as the active revocation because the amount of delayed updates should be small for such the short period.

On the other hand, delaying the reencryption process and disk I/O to some idle period of the CPU and disk, respectively, can utilize system resources more effectively to improve performance, multiple updates for the same portion can be gathered as one update. This is similar to the concept of lazy revocation but does not have the unsafe period of lazy revocation. Considering the reliability aspect, if we adopt the single-fault assumption, we can tolerate a disk or a node failure by keeping the update data in volatile memory on the backup node. This is the same as the concept of the asynchronous nWAL protocol. Therefore, we evaluated the performance effect of the BA-Rev with the delayed reencryption and disk write, and estimated their reliability.

B. Related Work

SNAD[9], Plutus[10], and SiRiUS[11] are secure storage systems that adopt the encrypt-on-disk scheme. The SNAD paper describes the revocation method as an issue for future work because of the trade-off problem. Plutus adopts lazy revocation, while SiRiUS adopts active revocation. These systems assume an environment in which clients consider systems to be insecure (systems are not trusted), so they allow decryption only on client machines. Thus, their policy is different from ours, because we implement reencryption on storage for efficiency reasons. Maat[12] uses automatic

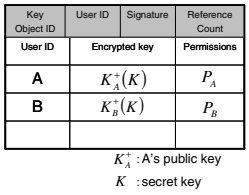

Figure 3. *key object*

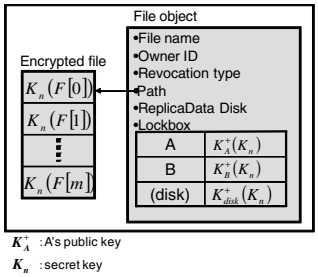

Figure 4. Data structure

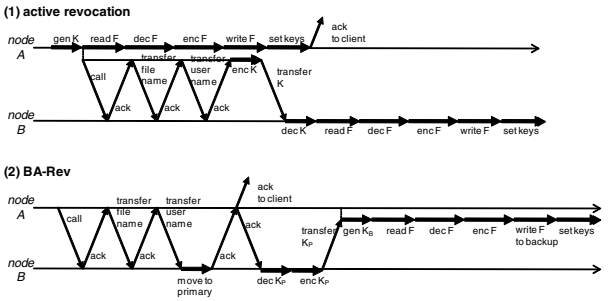

Figure 5. Process flows for a revocation

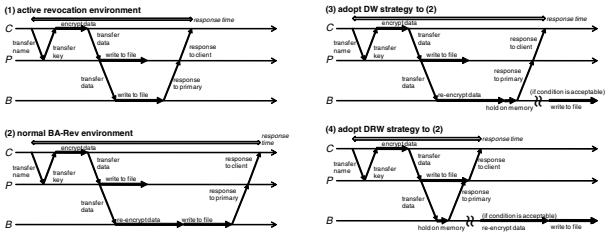

Figure 6. Processes of update in client (C), primary (P), and backup (B)

revocation in the Ceph file system, but they assume a very short lifetime. On the other hand, our approach is applicable to other systems if they adopt primary-backup configuration.

There are some cryptographic technologies for HDD in which all stored data are encrypted by a cryptography processor embedded in the HDD, such as "DriveTrust" by Seagate Technology and "MTZ2 CJ" by Fujitsu. These technologies encrypt all data on the HDD to protect the data if the HDD is lost, which is a different policy from data transmission protection. However, the cryptography processor is also useful for our approach.

C. Implementation Issues

Key management is important for systems using cryptography, and should be distributed if storage nodes are to meet the concept of fully distributed control. We place a lockbox which is proposed in SNAD [9] in each storage node, as a structure storing keys to allow authorized to obtain keys. Figure 3 shows an example of the lockbox, *Key object*. Here, a user x has a public key K_x^+ and a private key K_x^-, and a file is encrypted with a secret key K. Each column of *key object* comprises an authorized user's ID, a secret key encrypted with the authorized user's public key, and permissions. Only the authorized user can obtain the secret key because the authorized user's private key is necessary to decrypt the secret key. The file owner can distribute keys without communicating directly with sharers. In addition, storing secret keys so that they can be transmitted "as is" provides efficient key distribution.

We also use the data structure shown in Figure 4, based on SNAD. We assume that each user and storage node has a pair of public and private keys, (K_x^+, K_x^-), and that each file is encrypted with a secret key, K_n. At the time of encryption, the file is divided into one or more fixed blocks, and each block is separately encrypted to decrease the cost of update by defining the basic unit of updated data.

Each file object has information about the file name, the owner's ID, the path of the encrypted file, and a lockbox. Primary file objects also have information identifying the nodes where the backup data are stored. Lockboxes have keys not only for authorized users but also for the storage node in which they are stored because this node must implement reencryption processes. In addition, the lockboxes of backup files in the proposed method contain the primary decryption keys because the key used with the backup data must not be known, and so the updated data encrypted with the primary key are sent to backup when files are updated.

We selected the AES algorithm and ECB mode as the encryption mode. AES is considered to be robust against, for example, known plaintext attacks. ECB is the mode by which data are encrypted in independent blocks, and we used it to enable updates to be implemented by fixed-length blocks.

D. Process Flows

Process flows for active revocation with the primary–backup structure and BA-Rev are shown Figure 5, and update processes with DW and DRW strategies are shown in Figure 6. In the figure, each horizontal arrow indicates a process implemented in a node or an aspect of parallel processing in a node, depending on the situation. We omit disk I/O, file transmission, and index updates from the figure for simplicity.

Active Revocation:: Primary and backup data are reencrypted immediately. After generating a new secret key in node A, read, reencrypt, and write operations are implemented. The key, the file name and the revoked user's name are concurrently transmitted to node B via another thread, and the same process is implemented in node B. When the processes in node A are completed, notice of termination is transmitted to the requesting client. If accesses

to the object file are attempted before termination, they are blocked and made to wait until processes are completed.

BA-Rev:: After receiving a revoke request, node A transmits the information to node B and waits. Node B deletes the secret key for revoked users stored in the file object for backup data, sets the backup as the new primary data, and notifies process termination to node A. Because the new primary data are ready at this point, node A notifies termination to the client. If an access request to the object is received before this time, the system makes it wait and returns a request to reaccess together with information about the location of the data after process completion. Next, node A receives the key for the new primary data from node B and stores it in the lockbox of the new backup data for use when the file is updated. Finally, the new backup data are reencrypted with a new key and set as the backup in node A.

DW Strategy:: In the update process in backup, the process of writing the updated data (which are already reencrypted with the key used to the backup data) to the file is delayed. Unapplied updated data are kept in memory.

DRW Strategy:: The processes of reencrypting and writing updated data are delayed, and they are also kept in memory.

We consider four timings to write the unapplied updated data kept on memory into the backup disk in DW and DRW strategies:

T1: when a revocation occurs,
T2: when the amount of unapplied updated data exceeds a threshold (to avoid exceeding the limit of memory resources), and
T3: when one of the following conditions are satisfied
  T3a: after a constant period of time, and
  T3b: when the load is less than a threshold.

V. Experiments

We have performed experiments to evaluate the performance of BA-Rev with the DW and DRW strategies for the following cases.

i) **Active Revocation:** When a revocation occurs and the primary and backup data are encrypted with the same key.
ii) **BA-Rev:** When a revocation occurs and the primary and backup data are encrypted with different keys.
iii) **BA-Rev +** $\mathrm{DW}_{\mathrm{raw}}$/$\mathrm{DRW}_{\mathrm{raw}}$**:** In addition to ii), the DW and DRW strategies are applied to update processes. The timing for their application is T1 or T2 described in the previous section. Each storage node checks the amount of unapplied updated data at a constant interval and applies the updates if the amount exceeds the threshold.
iv) **BA-Rev +** $\mathrm{DW}_{\mathrm{const}:n}$/$\mathrm{DRW}_{\mathrm{const}:n}$**:** In addition to iii), the timing T3a is used. Defining n as a control parameter, the node waits to apply the updated data for n seconds after an update occurs, and after that, applies the update. If another update request for the same file is performed within n seconds, the new data replace or augment the updated data kept in memory.
v) **BA-Rev +** $\mathrm{DW}_{\mathrm{load}:n}$/$\mathrm{DRW}_{\mathrm{load}:n}$**:** In addition to iii), the timing T3b is used. Defining n as a control parameter, each node checks the number of active threads at a constant interval, then applies the unapplied updated data if the number is lower than n. This process is repeated until the condition is fulfilled.

Table I
Configuration of experimental system

CPU	AMD Athlon XP-M1800+ (1.53 GHz)
Memory	PC2100 DDR SDRAM 1 GB
HDD	TOSHIBA MK3019GAX (30 GB, 5400 rpm, B 2.5 inch)
Network	TCP/IP + 1000BASE-T
OS	Linux 2.4.20
Java VM	Sun J2SE SDK 1.5.0_03 Server VM

A. Experimental Environment

We developed a client–server program running on a PC cluster with the specification listed in Table I. Table II shows the parameters used in the experiments. The initial allocation of data among the servers follows chained declustering [2], in which backup data are stored in a neighboring node of the server storing the primary data. The client program sends requests to GET or UPDATE a file and measures response times for the requests. The definition of the response time for UPDATE for each method is shown in Figure 6. The client also sends requests to revoke or authorize a user's access permission for a file.

The interval for GET or UPDATE requests is determined by an exponential distribution $f(t) = \lambda e^{-\lambda t}$ in which the average arrival rate is $1/\lambda$. The request type is selected in accordance with the GET:UPDATE ratio, which is fixed at 50:50 in this paper. The objects of GET and UPDATE are selected from files stored in a corresponding storage node in accordance with a Zipf distribution [13] with parameter θ.

B. Response Times of Usual Accesses

We first evaluated response performance without revocation. We used three storage nodes, each of which initially stored 500 files of size 1 MB. A client node sent GET or UPDATE requests to a storage node, and nodes were selected with equal probability. We measured the average response time as the average arrival interval decreased from 500 ms to 150 ms.

The performance of GET was about the same in all cases because the process of GET was the same in all cases. Therefore, we omit the graphs of response times for GET.

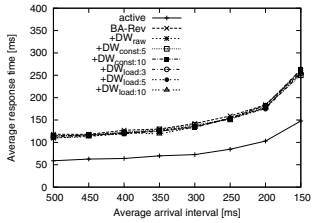

Figure 7. Average response times for UPDATE with DW strategy

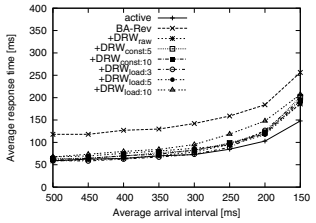

Figure 8. Average response times for UPDATE with DRW strategy

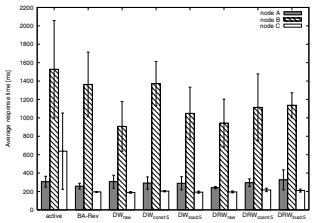

Figure 9. Average response times for concentrated revocations (GET)

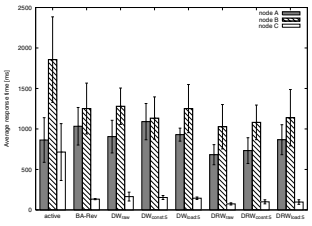

Figure 10. Average response times for concentrated revocations (UPDATE)

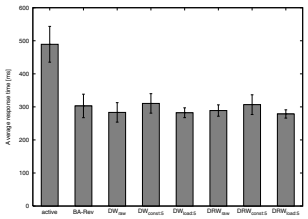

Figure 11. Average response times for distributed revocations (GET)

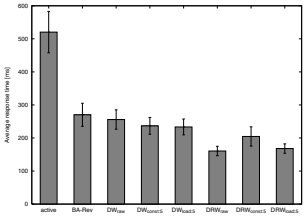

Figure 12. Average response times for distributed revocations (UPDATE)

Figures 7 and 8 show the average response times of UPDATE with the DW and DRW strategies, respectively. Comparing BA-Rev and active revocation, the response time of UPDATE with BA-Rev is longer: reencryption of the updated data for backup must be processed because different keys are used for primary and backup.

There was little benefit in applying the DW strategy to BA-Rev in all applied conditions of updated data, because the writing to disk is processed quickly by caching the updated data in this experiment. On the other hand, the DRW strategy significantly improved the UPDATE performance of BA-Rev, because the reencryption process is rather resource consuming. Comparing the performances in each case, although the performance in $\mathrm{DRW}_{\mathrm{load:10}}$ is slightly inferior because it includes the case in which multiple updated data are applied at high load, the performances in other cases are equivalent because there are both merits and demerits in each case.

C. Under Concentrated Revocations

To evaluate the influence of revocations, we measured response times when revocations occurred in a storage node. We used three storage nodes, *A*, *B*, and *C*, and three client nodes for this experiment. The backup data corresponding to the primary data stored in node *A* (*B*, *C*) were stored in node *B* (*C*, *A*). We fixed the average arrival interval of accesses at 400 ms, and the parameter *n* for DW or DRW was five. We enforced the multiple revocation processes for 50 files stored in node *B* simultaneously and measured response times of 100 GETs or UPDATEs, which were performed in the period containing the start and the end of the revocation processes. We performed this experiment six times.

Figures 9 and 10 show average response times and 95 percent confidence intervals for each storage node under the above conditions. With BA-Rev with or without DW and DRW, the response times of GET and UPDATE at node *B* storing target files of the revocation and at node *C*, which stores their backup data, are shorter than that for active revocation. This is because no reencryption for revocations was necessary in node *C* in BA-Rev, with or without DW and DRW. The reencryption was executed in node *B*, but the target of reencryption is backup data, which are not accessed from clients. On the other hand, at node *A*, response times for BA-Rev with or without DW and DRW are equivalent to, or slightly larger than, those for active revocation. This is for the reasons given in V-B. However, compared with the results at nodes *B* and *C*, the difference is small.

Comparing DW and DRW with BA-Rev, these strategies provide better GET performance, especially in node *B*. This is because they reduce the number of writes of updated data in the backup. For UPDATE, policy DRW, in which reencryption of updated data is delayed, has better performance than BA-Rev with DW. Thus, the reencryption process is susceptible to revocation processes, which also involve reencryption.

D. Under Distributed Revocation

To evaluate the mutual influence of multiple revocations occurring in contiguous storage nodes, we distributed the target files for revocation to all three storage nodes. We enforced multiple revocations for 15 files in each storage node and measured the response times of 100 accesses during the revocations. We performed this experiment six times.

Figures 11 and 12 show average response times and 95 percent confidence intervals for all storage nodes. Comparing BA-Rev, DW and DRW with active revocation, each proposed environment has better performance for both GET and UPDATE, similar to the results for the concentrated revocation examined above. In particular, the response time of UPDATE with DRW is the shortest, while those for other BA-Rev with or without DW and those of GET are nearly equivalent. The reason for the equivalent results is that the load per storage node is low in this experiment because of

Table III
DEFINITION OF PARAMETER FOR MTTDL

$MTBF_{Disk(Power)}$	MTBF of HDD, or power supply system
$MTTR_{Disk}$	MTTR of the HDD
N	number of storage nodes
M	maximum bandwidth of a storage node
R	bandwidth the node can allocate to repair
S	scale factor about the correlate disk failure
P	probability of successfully reading all disk sectors
E	the processing speed for re-encryption

the distributed revocation targets to multiple nodes. This result indicates that the BA-Rev realizes steadily better performance than that with active revocation, regardless of the method of applying updated data, if the number of revocation objects per node is not large.

VI. RELIABILITY ESTIMATION

With the DW and DRW strategies, data are temporarily inconsistent between the primary and backup disks because the updated data for backup are kept in memory. Therefore, there is the possibility of data loss from the combination of a disk failure and a power fault, even if the data are redundant because of the primary–backup structure. In this section, we estimate and compare Mean Time To Data Loss (MTTDL) of the normal environment adopting chained declustering (with active revocation or primitive BA-Rev) in the environment with DW or DRW.

A. Deriving MTTDL

We derived MTTDL using the method described in [14], and the parameters are defined in Table III. We use three factors for HDD failure: disk failure (DF), correlated disk failure (CDF), and unrecoverable bit error (UBE). For CDF, failures are commonly based on environmental factors (e.g., earthquakes, power dips and surges) and manufacturing factors (e.g., early or late failures in disk drives) [14], taking into account the higher probability of subsequent disk failures. With the UBE, the bit error rate (BER) is considered, this being the probability of encountering single bit errors.

B. Reliability with Chained Declustering

When a failure on storage node (k) occurs in the environment with chained declustering, the neighboring node ($k+1$) must set the backup data as the new primary, receive the data from node ($k-1$) for the backups lost on node k, and send the new primary to node ($k+2$) to repair the redundancy. Therefore, the amount of data to be transferred is the disk size at worst, so $MTTR_{Disk}$ is as follows.

$$MTTR_{Disk} = \frac{DiskSize}{M \times R \times 3600}[hrs] \tag{1}$$

MTTDL can be calculated for each factor of failure independently. With chained declustering, data are lost if one storage node has failed and one of the two neighboring nodes also fails before the repair has completed. Each MTTDL is therefore calculated as follows, and the overall $MTTDL_{Disk}$ is calculated as the harmonic average of the individual MTTDLs.

$$MTTDL_{DF} = \frac{MTBF_{Disk}}{N} \times \frac{1}{\frac{MTTR_{Disk}}{MTBF_{Disk}/2}} \tag{2}$$

$$MTTDL_{CDF} = \frac{MTBF_{Disk}}{N} \times \frac{1}{\frac{MTTR_{Disk}}{MTBF_{Disk}/S/2}} \tag{3}$$

$$MTTDL_{UBE} = \frac{MTBF_{Disk}}{N} \times \frac{1}{1-P^2}, P = \left(1 - \frac{1}{\frac{BER}{4096}}\right)^{\lceil \frac{DiskSize}{512} \rceil} \tag{4}$$

C. Reliability with DW and DRW

For the DW or DRW strategies, there are two combinations of failures that cause loss of data: one is a combination of storage nodes failing, and the other is a combination of a single node failure and a power failure. We deal with these separately.

1) MTTDL for Storage Node Failures: The calculation method is the same as in VI-B, except $MTTR_{Disk}$. With the DW or DRW strategies, when backup data are set as primary to repair, the updated data that are unapplied must be applied. Therefore, $MTTR_{Disk}$ is longer (because of the writing of the updated data in the DW strategy, and because of the reencrypting and writing of them in the DRW strategy) than defined in formula 1.

2) MTTDL for Storage Node and Power Failure: When a power supply fails, the updated data kept in memory are lost. Therefore, a storage node must send the primary data to the neighboring node to repair the backup data. About a disk's quantity of data must be transferred per node in the worst case, so the formula for $MTTR_{Power}$ is the same as formula 1.

There are two patterns in which data are lost with power supply failure: when the power supply fails and any disk error follows before the repair is completed ($P \rightarrow \{DF, CDF, UBE\}$); and when any single node fails and a power failure follows before the repair ($D \rightarrow P$). In the latter case, strictly speaking, data are not lost if the updated data have already been applied, but we assume that this situation occurs negligibly often in this paper.

Each MTTDL is as follows, and $MTTDL_{Power}$ is also calculated by the harmonic average of them.

$$MTTDL_{P \rightarrow DF} = MTBF_{Power} \times \frac{1}{\frac{MTTR_{Power}}{MTBF_{Disk}/N}} \tag{5}$$

$$MTTDL_{P \rightarrow CDF} = MTBF_{Power} \times \frac{1}{\frac{MTTR_{Power}}{MTBF_{Disk}/S/N}} \tag{6}$$

$$MTTDL_{P \rightarrow UBE} = MTBF_{Power} \times \frac{1}{1-P^N} \tag{7}$$

$$MTTDL_{D \rightarrow P} = \frac{MTBF_{Disk}}{N} \times \frac{1}{\frac{MTTR_{Disk}}{MTBF_{Power}}} \tag{8}$$

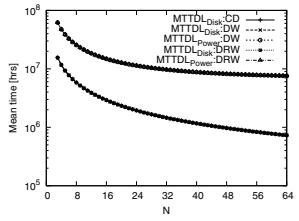

Figure 13. The MTTDL of disk failure in each environment and of power failure with DW/DRW

Table IV
ESTIMATION PARAMETERS

Disk size	500 GB
$MTBF_{Disk}$	$1.2 \times 10^6 hrs$
$MTBF_{Power}$	$1.752 \times 10^6 hrs$
BER	10^{-14}
M	300 MB/s
R	0.5
S	10
E	2 MB/s

D. Discussion

We compared MTTDLs in the environment with the chained declustering (CD) and that with DW or DRW. The parameters for estimation are listed in Table IV, referring to [14], [15] to select these values.

Figure 13 shows the MTTDL for disk failures in each environment and that with power failure with the DW or DRW strategies. The lines of CD, DW, and DRW of MTTDL for disk failures ($MTTDL_{Disk}$) are almost overlapped, and the difference is smaller than one percent of MTTDLs. In addition, $MTTDL_{Power}$, which expresses the probability of data loss caused by power supply failure in the environment with DW or DRW, is much larger than $MTTDL_{Disk}$. For example, in the case of $N = 32$, $MTTDL_{Power}$ is about 10^7 hours while $MTTDL_{Disk}$ is about 10^6 hours for the given parameters. Therefore, we can say that reliability degradation with DW and DRW is very small.

VII. CONCLUSIONS

We have proposed backup-assisted revocation (BA-Rev) with delayed write (DW), and with delayed reencryption and write (DRW). BA-Rev is a more efficient reencryption method for revocation in the encrypt-on-disk system than the active revocation method, and more secure than lazy revocation. By substituting the backup data encrypted with a key different from that of the primary for the primary data when a revocation occurs, the revocation is completed rapidly at low cost. To improve the update performance decreased by the backup handling, the reencryption process and disk write on the backup node are delayed, which has no ill effect on security and a very small decrease in reliability.

The experimental results using a PC cluster show that the proposed approach greatly reduces the average response times of accesses when revocations occur compared with active revocation. In addition, DRW achieves update performance equivalent to active revocation, and improved revocation performance. The estimation of reliability shows that the decrease in MTTDL by the DW and DRW is smaller than one percent of MTTDL for the normal chained declustering. Those results demonstrates that the proposed approach provides high performance and availability for secure network storage systems.

In future work, we plan to evaluate the proposed approach in actual environments including different types and sizes of files accessed from heterogeneous applications.

ACKNOWLEDGMENTS

This research was partially sponsored by CREST of JST (Japan Science and Technology Agency) and MEXT, via a Grant-in-Aid for Scientific Research #19024028.

REFERENCES

[1] Brian Hayes. Cloud Computing. *CACM*, 51(7):9–11, 2008.

[2] Hui-I Hsiao and David J. DeWitt. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In *Proc. of ICDE'90*, pages 456–465, 1990.

[3] Paul Stanton. Securing Data in Storage: A Review of Current Research. *ArXiv Computer Science e-prints*, 2004.

[4] Erik Riedel, Mahesh Kallahalla, and Ram Swaminathan. A framework for evaluating storage system security. In *Proc. of FAST '02*, pages 15–30, 2002.

[5] Kazuki Takayama. Compatibility of performacne and security in distribute storage system storing encrypted data (in japanese). Master's thesis, Dep. of Computer Science, Tokyo Institute of Technology, 2009.

[6] Svein-Olaf Hvasshovd. *Recovery in Parallel Database Systems*. 1996.

[7] Haruo Yokota. Autonomous Disks for Advanced Database Applications. In *Proc. of DANTE'99*, pages 435–442, 1999.

[8] Haruo Yokota, Yasuhiko Kanemasa, and Jun Miyazaki. Fat-Btree: An Update-Conscious Parallel Directory Structure. In *Proc. of ICDE1999*, pages 448–457, 1999.

[9] Ethan Miller, Darrell Long, William Freeman, and Benjamin Reed. Strong Security for Network-Attached Storage. In *Proc. FAST '02*, pages 1–13, 2002.

[10] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In *Proc. of FAST'03*, pages 29–42, 2003.

[11] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. SiRiUS: Securing Remote Untrusted Storage. In *Proc. of ISOC and NDSS Sympo. 2003*, 2003.

[12] Andrew W. Leung, Ethan L. Miller, and Stephanie Jones. Scalable Security for Petascale Parallel File Systems. In *Proc. of SC07*, pages 1–12, 2007.

[13] Donald E. Knuth. *Sorting and Searching*. Addison-Wesley Publishing Company, 1973.

[14] Edward Michael Macdonald. Designing Reliable Large-Scale Storage Arrays. Master's thesis, Florida State University, College of Engineering, 2007.

[15] Frank Bodi. "DC-Grade" Reliability for UPS in Telecommunications Data Centers. In *Proc. of 29th INTELEC*, pages 595–602, 2007.