Intermittent Snapshot Method for Data Synchronization to Cloud Storage

Yuichi Yagawa, Mitsuo Hayasaka, Nobuhiro Maki, Shin Tezuka, Tomohiro Murata

Abstract—**We developed a configuration to place a cash storage called Cloud on-Ramp (CoR) with small capacity between client terminals at remote offices and a cloud storage connected by a wide area network. A CoR is placed in each remote office, and applications access them via a local area network, guaranteeing access performance. A CoR synchronizes the data with the cloud storage by copying updated data from the client terminal at regular intervals. It is necessary that a window to carry out the bulk-copy function be shortened and repeated many times to keep the latest data in the cloud storage. A method of maintaining consistency of data at a CoR is necessary, even if the data are updated in the copy by the client terminal. Therefore, we propose the "intermittent snapshot method", with which a snapshot is taken during bulk-copy execution and released as soon as the bulk copy is over. We evaluated the proposed method from the perspective of an implementation design. We formalized the method by using a stochastic Petri-net model and considered the proper size of the bulk-copy window that optimizes both the synchronization delay and application-access performance through simulation.**

***Index Terms*— Cloud Storage, Cloud on-Ramp (CoR), Data Synchronization, Intermittent Snapshot, Stochastic Petri-net**

I. Introduction

Cloud storage [1][2] has begun to be used for collaboration between remote offices in a company. The configuration to place a cash storage with small capacity between a client terminal and cloud storage has been suggested because the problem of delay arises from the client terminal in a distributed environment connected to the cloud storage through a wide area network (WAN) [3]–[5]. A cash storage is placed in each remote office, and applications access them via a local area network (LAN), guaranteeing access performance. We call such a cash storage Cloud on-Ramp (CoR) [3]–[5].

A CoR synchronizes the data with the cloud storage by copying updated data from the client terminal to the cloud storage based on the write-back cash algorithm in a lump at regular intervals. We call this a "bulk-copy" function. This function should not affect the access performance from the client terminals and their applications. Therefore, this function is executed when work in the remote office usually finishes, including nighttime.

The demand of placing the latest data into the cloud storage has recently increased. For example, there is a need of sharing the data that have accumulated in many CoRs at remote offices and of referring them through the cloud storage immediately. It is also necessary to place as much of the latest data into the cloud storage as possible because the data need to be recovered from the cloud storage at the time of data access errors in a CoR. We call the difference in the synchronization time in the cloud storage determined from a CoR "synchronization delay". The shorter the synchronization delay, the fresher the data users can access through the cloud storage.

It is necessary that the distance to execute the bulk-copy function be shortened and repeated many times to shorten the synchronization delay. We call this distance the “bulk-copy window". However, the time required to execute bulk copy is prolonged for the time it takes all copies of the data to update because a re-copy runs when a file is updated in the bulk-copy window [6].

A method of maintaining consistency of data is necessary even if the data are updated in the copy by the client terminal to shorten the bulk-copy window and repeated many times. Generally, a method of maintaining the consistency of data copying includes a snapshot method [7]. In a CoR, a snapshot of the update data is taken from the client terminal and copied to the cloud storage. In this study, we assumed office applications and data processing applications through the sharing of data among remote offices, but there are more read commands (reading of data from a CoR) than write commands (writing data to a CoR) in the input/output (IO) of those applications. Therefore, we decided to adopt the Copy On Write snapshot method (COW), with which the effect on read performance is zero [7].

However, there is problem with COW in which the effective performance of the write judgment from the applications degrades. Therefore, we propose the "intermittent snapshot method", with which a snapshot is taken during the bulk-copy execution and released as soon as the bulk copy is over. The performance penalty in the write from the applications is reduced by limiting the time to take the snapshot.

There is a problem in the implementation to appropriately determine the bulk-copy window. Users request to shorten the window and place the latest data into the cloud storage. However, the number of bulk-copy execution times increases; in other words, it becomes easy for the applications to incur a performance penalty due to the increased time of snapshots to

Yuichi Yagawa, Mitsuo Hayasaka, Nobuhiro Maki and Shin Tezuka are with the Hitachi, Ltd., Research & Development Group, Kokubunji-shi, Tokyo 185-8601, Japan (corresponding author to provide phone: +81-42-327-7885; fax: +81-42-327-7687;
e-mail: yuichi.yagawa.bh@hitachi.com, mitsuo.hayasaka.hu@hitachi.com, nobuhiro.maki.vg@hitachi.com, shin.tezuka.xs@hitachi.com).

Tomohiro Murata is with Graduate School of Information, Production and Systems, Waseda University, Kitakyusyu-shi, Fukuoka 808-0315. (e-mail: t-murata@waseda.jp).

ISBN: 978-988-14047-8-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

be taken. There are also cases in which a bulk-copy window increases when the preceding bulk copy-function is not finished before the next one is started. Conventionally, this bulk-copy window was determined from experience.

For this technical problem, we formalize our intermittent snapshot method in the stochastic Petri-net model and calculate a suitable bulk-copy window through simulation.

II. Bulk-copy function and its problem in CoR

A. System architecture

In this chapter, we give an overview of and the problem with the bulk-copy function in a CoR.

The system architecture we assumed for this study is based on that of a CoR that we previously proposed [3]–[5]. As shown in Figure 1, the system is composed of CoRs placed in remote offices, applications in client terminals accessing the CoRs, and a cloud storage connected from the CoRs.

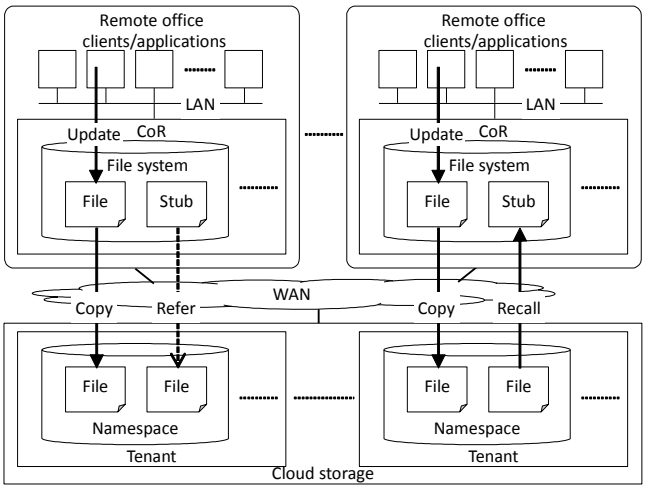

Fig. 1. Assumed System Architecture

End users operate the application programs in the client terminals, which generate file data. A write request to store the data in a CoR and a read request to obtain the data that have been stored away by the CoR are published. The request is made in general file-sharing protocols such as CIFS or NFS. In addition, the client terminals are connected to a CoR by high-speed LAN.

A CoR is a cash storage that temporarily stores file data. The request received from the client terminals are handled, and data are stored or provided. Furthermore, the bulk of data is regularly copied to a cloud storage to protect the stored data. Regarding the file data on a CoR where bulk copying ended, the use situation is checked regularly. Because a CoR has small capacity, the data with a small access frequency leave only a reference to the replicated data written in the cloud storage, and the CoR deletes those data. When a read request for the data is received, the data are first recovered from the cloud storage to the CoR then returned to the client terminals. Please refer to a previous study [5] regarding this process.

A cloud storage is an object storage that permanently stores file data. A replication request received from a CoR is handled, and data are stored. The HTTP/HTTPS suitable for communication in a wide area is used for the request. The data are reproduced at several inter-nodes constituting the cloud storage and are protected. In addition, the network to connect CoRs and the cloud storage is assumed to be a WAN, which is slower than a LAN, because such a network is used with a system that is applied to remote offices located over a wide geographic area.

The bulk-copy function by a CoR is regularly executed based on the bulk-copy window. A system manager can set the bulk-copy window, which is determined depending on the demand from the end users. For example, the window is kept short if the end users want to keep the latest data shared in the cloud storage.

B. Bulk-copy function

The bulk-copy function is composed of the following three processes.

In the pre-process, the data for the bulk-copy function are extracted. A CoR sorts updated data targeted for existence confirmation, a copy of a file, and directory and outputs a copy-entity list. Therefore, the CoR records the operation history of the client for a file and directory using the operating system function beforehand [8].

In the replication process, the CoR transfers the data of a file and the directory listed in the copy-entity list to a cloud storage. The cloud storage receives the data and stores it.

In the post-process, the result of every bulk-copy function being executed repeatedly is recorded, for example, when a process is not finished in a bulk-copy window when the data for a large quantity of copies exist. In this case, the current iteration continues and the next iteration is skipped, but the results are recorded. It will be confirmed whether the manager can keep the bulk-copy window the end users require with reference to this record.

It is necessary that the bulk-copy function be executed to maintain the data consistency of the IO requests from applications. Therefore, a snapshot method is used. The applications access a primary volume, and the bulk-copy function accesses a snapshot volume while maintaining data consistency.

C. Conventional snapshot methods and their challenges

There are generally two methods of taking a snapshot, i.e., redirection on write method (ROW) and COW [7]. The overview of the two methods and the challenges when applied to the bulk copy are explained below.

Redirection On Write snapshot method (ROW)

ROW is a snapshot method using log-structured data management. Every chunk of new data is added to a physical volume as the log structure, and a logical address refers to the address in the physical volume. The chunk is first written to add a postscript to the physical volume when an application writes in a file and directory (postscript data). The primary volume manages the physical address of each chunk logically in B+tree. The snapshot volume manages the snapshot equally as an aggregate of the physical address. The snapshot is converted into a data array that should be read with postscript data by converting the logic address of the chunk the application requires into a physical address.

When ROW is applied to the bulk-copy function, there are

ISBN: 978-988-14047-8-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

problems in performance. In both primary and snapshot volumes, performance always degrades regarding the address translation of the chunk at the time of the read. Particularly, it is easy to follow that the data of one file are located at the far-off position on the physical disk, and the performance penalty at the time of the read is large. The write performance to the primary volume also degrades. Because address translation and chunk allocation must be assigned regardless of the write size, write performance always degrades.

Copy On Write snapshot method (COW)

COW is implemented as a function of the logical volume manager (LVM) and takes a snapshot by using the snapshot volume and COW table for a reference prepared for separately from the primary volume. When a client overwrites with a file and directory, the LVM first detects a chunk of data to be overwritten and copies it in the snapshot volume. Furthermore, an address of the copy is recorded on the COW table. COW overwrites with new data in the overwrite chunk afterwards, and the COW table is stored in the fixed position of the snapshot volume.

There are also problems in performance when COW is applied to the bulk-copy function. Write performance to the primary volume largely decreases at the time of taking a snapshot (snapshot period). This is because the number of reading and writing requests of the volume increases 3-fold, compared with the normal IO. In addition to the write of the new chunk, the read of the chunk to be overwritten, write of the chunk to the snapshot volume, and write of the COW table occur. In addition, the read performance of the snapshot degrades. This is because the data in the snapshot are detected from the COW table and searched from the snapshot volume. On the other hand, the read performance of the primary volume by the application does not degrade.

D.Problems in implementing bulk-copy function

As explained above, ROW and COW cause problems in read/write performance. The frequency of the bulk-copy function increases and it is expected that the synchronization delay is shortened to reflect the latest data in the cloud storage. However, it becomes easy for an application to incur a penalty in read or write performance degradation due to implementing the conventional snapshot methods because the snapshot period increases. Also, a bulk-copy window increases when the processing of the bulk-copy function in front does not finish before the next one is started.

In contrast, choosing the appropriate snapshot method in the assumed use environment and the appropriate copy-window size (time width) are systematic problems. However, the bulk-copy window size is conventionally determined from experience and an appropriate bulk-copy window size and performance limit of the bulk-copy function based on the size has not been studied.

III. INTERMITTENT SNAPSHOT METHOD

We developed our intermittent snapshot method to limit the snapshot period. In other words, a snapshot starts to be taken just before the replication handling of net-net start, i.e., a file data transfer, and finishes just after the replication process ends. This makes it possible to mitigate the performance degradation in the remaining time of the bulk-copy window after the snapshot. This process is repeated at every bulk-copy function. General office applications and data-processing applications are assumed through the data sharing among remote offices, and COW is used to determine the effect on read performance, which is zero in consideration that there are more read requests than write requests in the IO of such applications.

A. Overview of intermittent snapshot method

This method involves the bulk-copy function, write process from applications with taking a snapshot, and normal write process without taking a snapshot. Though a bulk-copy function is executed repeatedly, these processes are executed alternately, as explained in the following paragraph. In addition, other processes including read from applications are not changed from conventional processes.

In the bulk-copy function, the CoR repeatedly replicates collective file data in the copy-entity list to the cloud storage. These file data are received in the cloud storage and stored.

At replication start, a snapshot starts to be taken first. Then, the write process from applications with taking a snapshot replaces the normal write process. When a write request occurs, the CoR begins to read the current data and writes in the current data at the snapshot volume and renews the COW table. The new data are then written in a primary volume. At this time, the penalty for one read and three writes occurs in comparison with a normal write.

At replication end, the snapshot stops being taken. Then, the normal write process replaces the write process with taking the snapshot. When a write request occurs from the application, the CoR writes in the write request data at the primary volume.

The impact on application-write performance can be reduced by limiting the time to take the snapshot. The method is particularly effective with applications mainly composed of reads because the effect on read performance is zero with COW. It is also possible to reduce the cost of the cash storage because only the domain of the snapshot for one generation is needed in the CoR, whereas the snapshot domain of many generations is always necessary in the ordinary network-attached storage due to backup.

B. Design criteria for implementing intermittent snapshot method

With the proposed method, the relationship between the size of the bulk-copy window and synchronization delay needs to be addressed. For example, each bulk-copy function is started every hour or 30 minutes when a bulk-copy window is set for one hour or 30 minutes. In addition, each snapshot starts to be taken in sync with the bulk-copy function and completed when all data replications in the copy-entity list are finished. This snapshot period is the same as the bulk-copy-execution time, which expresses the synchronization delay, and must be shorter than the bulk-copy window. Particularly, it is expected that the snapshot period is further shortened because the performance penalty of write requests from applications to a CoR occurs with COW. This depends on the number of data replications and WAN transfer bandwidth, and the number of transfers is based on the number of files updated by the

ISBN: 978-988-14047-8-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

applications and their data size during the bulk-copy window in front.

The synchronization delay is expected to decrease because the latest data should be placed in the cloud storage. Therefore, it is necessary to shorten the bulk-copy window, but the number of starts of the bulk-copy function increases at the same time. As a result, it becomes easy for the applications to incur a performance penalty because the snapshot period increases. In addition, a bulk-copy window promised for the end users cannot be protected when the current bulk-copy function does not complete before the next one begins.

In a design implementation, it becomes necessary to appropriately determine how much the bulk-copy window can be shortened regarding the statistical frequency of read and write requests under the assumed application environment. Specifically, under the assumed write performance and WAN-transfer bandwidth, the smallest bulk-copy window, which is 100% of snapshots completed, will be necessary.

IV. Implementation Design

For the design criteria explained in Section III, we discuss a model of our intermittent snapshot method and a simulation experiment.

A. Stochastic Petri-net model of intermittent snapshot method

We first formalize our intermittent snapshot method using the stochastic Petri-net model to solve an implementation problem because the structure and behavior of a system can be expressed visually. Also, the stochastic Petri-net model can express random characteristics of write requests from applications and uncertainty of WAN data transmission.

The stochastic Petri-net model consists of a main process and a timing control, as shown in Figure 2. Each component is explained as follows.

Main process

In applications, stochastic transition T0 expresses write requests and publishes a write-request token based on probability distribution. It is assumed that the random nature of the write requests obeys an exponential distribution. Place P0 expresses divergence to T1 expressing the normal write processing without taking a snapshot and to T12 expressing the write processing with taking a snapshot.

In a CoR, T1 and T12 are connected from P6 by a permission arc or inhibitor arc respectively, then either T1 or T12 will be chosen based on the condition of P6. Transition T1 requires the processing time that is equal to normal write-request performance. Transition T12 requires the processing time based on the performance penalty of a COW snapshot. In addition, the write-request tokens from T1 or T12 are collected in P1.

Place P2 expresses a wait buffer for a bulk-copy function from a CoR to a cloud storage, and T2 controls the flow of tokens from P1 to P2. Transition T2 is connected by an inhibitor arc from P7 that expresses a state of a snapshot under the bulk-copy execution, and this transition is inhibited during the bulk-copy execution time. When the bulk-copy execution is completed (i.e. all data replications complete), T2 fires, and the write-request tokens that have become the candidates of the next bulk copy will be collected in P2.

In T3, the write-request tokens to the same files are adjusted. The mean probability of unique files updated among all write requests is denoted as *X*. Then, the number of unique files is multiplied by a data-set conversion ratio *Y* and converted into data-set tokens that will be replicated. Here, *Y* is calculated in Formula (1). The value necessary for calculating *X* and *Y* to use in simulation is determined from a real environment. In addition, T3 is connected in a permission arc by P7, so T3 fires during the bulk-copy execution and the data-set tokens for the bulk-copy execution will be collected in P3.

$$Y = \textit{average file size} \div \textit{data-set size} \quad (1)$$

In the cloud storage, stochastic transition T4 expresses WAN transfer delay, and its condition is the same as that in our previous study [5]. Place P4 is where all data-set tokens have been transferred in the cloud storage.

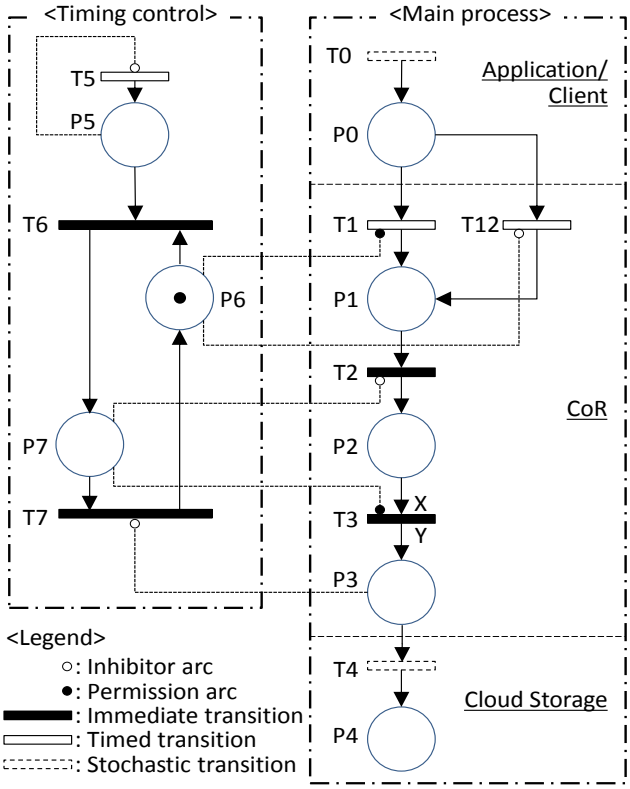

Fig. 2: Stochastic Petri-net model of proposed intermittent snapshot method

Timing control

Transitions T5–T7 and P5–P9 control each timing of the main process by delivering a token to each.

Time transition T5 expresses a bulk-copy window and publishes a token with a regular period of the window. Place P5 expresses waiting for completion of the bulk-copy function. If the previous bulk-copy function is not completed, a token remains in P5 and inhibits publishing by the token from T5. Transition T6 expresses a bulk-copy start when it fires. Place P6 expresses the state of normal write, permits T1 and inhibits T12 when there is a token, and inhibits T1 and permits T12 when there is no token. When there is a token in P5 and P6, T6 fires, and each token is removed from P5 and

ISBN: 978-988-14047-8-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

P6. Then, T12 fires, and taking a snapshot is started. (T1 is inhibited, and the normal write is terminated.)

Transition T7 expresses the bulk-copy completion when it fires, and it is connected to P3 by an inhibitor arc. In other words, T7 does not fire unless all transfer tokens of P3 disappear. Place P7 expresses a state of the bulk-copy execution, and a token will wait until all transfer tokens of P3 disappear and T7 fires. Transition T2 is inhibited in the state with a token in P7, and write-request tokens are accumulated in P1. On the contrary, T3 is permitted to fire in the state with the token in P7. All tokens in P2 are transferred to P3 for replication.

By using ROW or COW all the time, the stochastic Petri-net model has a configuration without T1 and the permission and inhibitor arcs from P6. The write-request tokens always go through T12 with a snapshot penalty. The other configuration of these conventional methods is the same as that in Figure 2.

B. Simulation experiments and results

We developed a simulator based on the model in Figure 2. With the simulator, the synchronization delay was measured while changing the bulk-copy window (time transition T5) to evaluate a suitable bulk-copy window size with the intermittent snapshot method.

Experimental conditions

The control settings of the experiment were as follows. We first measured the mean file size, mean file update rate, and mean write-request performance from the actual system in our workplace. The measurement period was from 8:00 to 20:00 on weekdays from June 22nd through September 5th, 2017. The mean file size was 1 MB, mean file update rate (X) was 33% (the rate except the write requests issued to the same file), and mean write-request performance (T0) was 6 operations per second (OPS), (the number of write requests issued for 381 OPS of all IO requests). We also assumed that the write requests (T0) occurred at random and in exponential distribution.

Regarding the WAN-data-transmission delay, stochastic transition T4 set a parameter by Pareto distribution based on a previous study published in 2005 [9]. We assumed that the WAN-data-transmission delay had not changed since 2005, so we used the same average and dispersion as that study.

However, the bandwidth of the WAN varies according to use environment. Therefore, we assumed that the data-set size in Formula (1) was 36 times larger than 9 KB, which was used in 2005, based on our environment. The Y was calculated as 3 tokens.

Each delay time of a normal write (T1) and a write with a COW snapshot (T12) was set to 8 and 17 ms respectively, as determined in a previous study [7]. Note that we assumed the delay time under the condition of a 100% write-request ratio in the previous study [7].

Applications were assumed to run from 8:00 to 20:00, and the measurement time was set to 12 hours. The bulk-copy window size was changed from 1 second to 15360 seconds. The number of simulation executions was 10, and we calculated the average of all results in the simulation executions.

Calculations in simulation

We investigated the relationship between the size of the bulk-copy window and synchronization delay. Specifically, we made sure of how much the bulk-copy window can be shortened under the condition in which the effectiveness for the write-request and 100% copy-success ratio to finish the bulk-copy execution in the copy window is maintained. The following formulas were calculated using the stochastic Petri-net model shown in Figure 2.

$$\textit{Effective performance of application-write requests (\%)} = (\textit{T1 number of fire times} + \textit{T12 number of fire times}) \div \textit{T0 number of fire times} \times 100 \quad (2)$$

All tokens fired by T0 should go through either T1 or T12 within the measurement time, and no tokens should remain in P0.

$$\textit{Copy-success ratio (\%)} = (1 - \textit{Number of inhibited times in T5}) \div \textit{assumed number of bulk-copy-execution times} \times 100 \quad (3)$$

Transition T5 should fire regularly to meet the bulk-copy window. However, T5 cannot fire if there is a token left in P5.

$$\textit{Synchronization delay} = \Sigma(\textit{T7 fire time} - \textit{T5 fire time}) \div \textit{T7 number of fire times} \quad (4)$$

The fire of T5 means the start of taking a snapshot as well as executing the bulk-copy function, and the fire of T7 means end of taking a snapshot. We assume that T5 and T6 fire at the same time if there is a token in P6 when the regular fire time is in time transition T5.

Experimental results

As a result, the effective performance *of application-write requests* was maintained at 100% even when the bulk-copy window was changed from 1 second to 15360 seconds. Also, the copy-success ratio was maintained as 100%, except the case in which the bulk-copy window was 1 second.

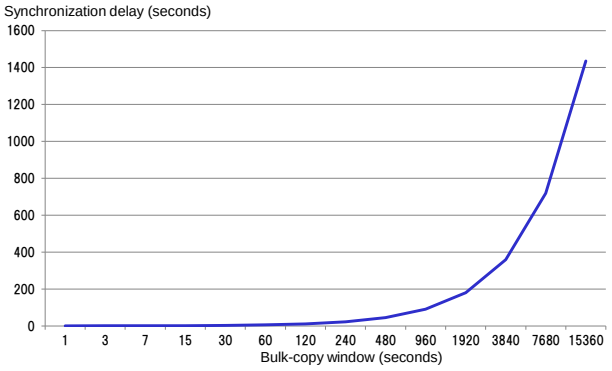

Fig. 3: Bulk-copy window and synchronization delay

The synchronization delay was measured, as shown in Figure 3. The synchronization delay rapidly increased from

ISBN: 978-988-14047-8-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

1920 seconds of the bulk-copy window, which indicates the limit of the bulk-copy window under given conditions. An appropriate bulk-copy window should be set below this limit. For example, if we choose 480 seconds as the bulk-copy window, the average synchronization delay becomes 45 seconds, which will meet end users' request in our workplace. However, further study is required to define an optimal bulk-copy window.

V. Conclusion

We proposed the intermittent snapshot method for a CoR to regularly synchronize data with cloud storage. The method adopts a COW snapshot to maintain consistency of data at a CoR, even if the data are updated in the copy by the client terminals, but limits the snapshot period. The snapshot is taken only during the bulk-copy execution and released as soon as the execution is over.

We also evaluated the proposed method from the perspective of an implementation design. It is necessary that the bulk-copy window be shortened and repeated many times to keep the latest data in cloud storage. We formalized the method by using the stochastic Petri-net model, and considered the proper size of bulk-copy window that optimizes both synchronization delay and application-access performance through simulation.

We also found that there is a limit of the bulk-copy window under given conditions. However, we need to consider the following points for future work.

The overheads of both the pre-processing and post-processing of the bulk-copy function may impact the overall throughput of the method. We assume that there is a tradeoff between synchronization delay and overhead, and that an optimal bulk-copy window can be determined from this tradeoff.

The intermittent snapshot method should be compared with the conventional methods of ROW and COW that always takes snapshot. We assume that our method should perform better from the applications perspective.

Acknowledgments

We thank Mr. Yoshiyuki Fujita and Mr. Tatsuya Matsumoto of Hitachi Ltd., IT Services Division for their support during the experiments.

References

[1] M. Alam and K. A. Shakil: "Recent Developments in Cloud Based Systems: State of Art," arXiv:1501.01323 (2015)

[2] V. Venkatesakumar, R. Yasotha and A. Subashini: "A brief survey on hybrid cloud storage and its applications," World Scientific News, Vol. 46, pp. 219-232 (2016)

[3] Y. Yagawa, A. Sutoh, K. Matsuzawa, Y. Fujita, O. Matsuo, and T. Murata: "A Cloud Storage Cache System to Improve Data Access Performance through WAN", The 29th International Technical Conference on Circuits/Systems, Computers and Communications (2014)

[4] Y. Yagawa, A. Sutoh, E. Malamura, T. Murata: "Modeling and Performance Evaluation of Cloud On-Ramp by Utilizing a Stochastic Petri-Net", 5th IIAI International Congress on Advanced Applied Informatics, pp. 995-1000 (2016)

[5] Y. Yagawa, A. Sutoh, E. Malamura, T. Murata: "Implementation Design and Performance Evaluation of Partial Recall Method", IEEJ Transactions on Electronics, Information and Systems, Vol. 137 No. 10 pp. 1414-1421 (2017)

[6] J. Nemoto, A. Sutoh, and M Iwasaki, "File System Backup to Object Storage for On-Demand Restore," Proc. 5th IIAI International Congress on Advanced Applied Informatics, pp. 946-952, Kumamoto, Japan, Jun. 2016.

[7] W. Xiao, Q. Yang, J. Ren, C. Xie, and H. Li: "Design and Analysis of Block-Level Snapshots for Data Protection and Recovery", IEEE Transaction on Computers, Vol. 58 (2009)

[8] M. Takata and Atsushi Sutoh, Event-notification-based Inactive File Search for Large-scale File systems, Proc. Asia-Pacific Magnetic Recording Conference, TA-3, 2012."

[9] T. Kashima, S. X. Kato, T. Akiyama, K. Nozaki, Y. M. Matsumoto, and S. Shimojo, "A Method for the Estimation of Collective Communication Time Using Probability Distribution of Communication Latency in Grid Environment", Vol. 46, No. SIG16 (ACS12) (2005)