Long Distance Synchronization Method for In-Memory State Management

Arif Herusetyo Wicaksono[1] Kazuhide Aikoh[1]

Abstract: With the needs of protecting data consistency between multiple data centers for state management in transactional system, a long distance data synchronization technology is required, both for improving performance by distributed processing and for providing fail-over in the case of large-scale disaster. However, in existing method, maintaining data consistency causes network latency degradation for ensuring completion of synchronization, while maintaining network latency performance causes large number of data inconsistencies in the case of large-scale disaster, both of which are unacceptable. Therefore, we propose a synchronization method with two main characteristics: prioritization of transmission of identifier information along with transaction ordering information for faster processing, and confirmation of replication status prior to I/O operation. By applying the proposal to an in-memory KVS, we verified that the number of data inconsistencies caused by disaster can be reduced by 39% for 100 kB value size, without affecting the network latency performance. Finally, it can be estimated that the minimizing data inconsistencies while maintaining network latency performance can be achieved with our proposed long-distance synchronization method.

Keywords: In-memory KVS, DR, Long-distance synchronization

1. Introduction

The state management component in a transactional system requires low-latency to enable fast response during peak time. In-memory processing is often utilized in such transactional system due to its capability of providing low-latency response. One example is adopting in-memory Key-Value Store (KVS) for state management.

However, such state management component usually involves several data centers located in major metropolitan area to improve the quality of services. Additionally, to ensure business continuity in the event of a large scale disaster, such as earthquake, the backup of the data should be stored in different data centers. In such deployment, it is important to maintain strong consistency[1] guarantee to prevent conflicts between information contained in different sites. Strong consistency can be obtained by implementing distributed consensus algorithm such as Paxos[2] or Raft[3], within nodes of a cluster to agree on which action to take.

To satisfy these two requirements, a long-distance synchronization technology to ensure the consistency while maintaining the low-latency performance is required. However, it is difficult to design such technology due to the trade-off between consistency and latency. Ensuring strong consistency model requires multiple communications between the member nodes for status confirmation, which cause system latency to degrade. On the other hand, ensuring low latency response means minimizing the number of communication between member nodes, thus relaxing consistency guarantee.

In this research, we aim to design an approach for long-distance synchronization, which maintains the low-latency performance of the data store, while having the capability of preventing the occurrence of inconsistencies, especially for use with KVS with strong consistency model.

2. Existing Method for Long Distance Synchronization

In long distance synchronization there are two main approaches, asynchronous and synchronous replication. In this section, we will consider the characteristics of each approach.

Asynchronous replication[4] technique can satisfy the low-latency response requirement, but cannot satisfy the strong consistency guarantee. By transmitting the replication data after the local operation finished at the time independent of the local operation, low-latency response of the system is preserved. However, if system disruption occurs prior to completion of the data transmission to the backup server, conflicting information on the same key may occur.

Synchronous replication[4] works in the opposite way. It can satisfy the strong consistency guarantee, but cannot satisfy the low latency operation. A put operation in the main server will not be confirmed, until the same put operation is completed in the backup server. By confirming that a data is successfully written in both main and backup server before the response is given to the client, conflicting information may never occur. Even with the fail-over to the backup server, the complete data is available. However, the trade-off with this approach is that the time it takes to finish a put operation will grow significantly, due to the in-

[1] Hitachi, Ltd., Research & Development Group, Center for Technology Innovation-Information and Telecommunications

volvement of round-trip network communication from main to backup server.

There have been some approaches [5] [6] which attempt to combine synchronous and asynchronous replication. However, those approaches could not satisfy both strong consistency guarantee and low-latency response at the same time, because they require the replication to be finished, for the data to be useful, which technically fall into the category of synchronous replication.

Approaches limited to software could not satisfy both low-latency response and strong consistency guarantee at the same time. In this research, we implemented hardware solutions, specifically network infrastructure, in addition to software processing technique.

3. Long distance synchronization method for in-memory KVS

Our method for long distance synchronization has three main components, key-value pair replication, replication status confirmation, and cellular networking for datacenter. In this section, we will describe each component and discuss our design choices.

3.1 Transmission of replication data from main site

Contrary to the traditional approach, to satisfy the consistency protection requirement in two distantly located data centers, completion of key-value pairs replication are not required. Only the key information from a key-value pairs has to exist at the same time. Specifically, only the information that a certain key has changed is required. To maintain the strong consistency guarantee, complete fail-over is not required, only error detection capability is required.

Therefore, we separated the key from its value, and prioritized it for transmission. The key size for the key information is small, thus it can be transmitted with only a single or few packets. On the other hand, value size is bigger, and majority of the replication time is often the time it takes to replicate the value data.

Consequently, as the tradeoff for separation between key and its value, we introduce a new variable called transaction identifier. This transaction identifier is utilized to recombine a key back to its value in backup site, and also maintain the order of transaction as it happens in the main site to preserve consistency.

The corresponding value is transmitted along with key and its transaction number at times independent of the key transmission. Due to its bigger size, the value transmission packet will take longer to finish compared to the key-only packet. Once replication of the value for a key is finished, then the key and its corresponding value is matched by transaction identifier and moved to main data store in the backup site. **Fig. 1** provides the flowchart of replication in our method due to a put request from a client.

3.2 Confirmation of replication status in the case of fail-over

In our configuration, as long as main server is operational, backup server does not answer to read request from client.

If a read request is received by the backup server for any key and the main server is not available, backup server checks for the existence of the key in the pending data queue. If the key does not exist, there is not any unfinished replication for the requested key, which means read request is forwarded to the main data store and it can be processed normally. However, if it does exist in the pending data queue, the read request should be deferred until the replication is completed. If the replication cant be completed, then a data lost is detected, and further access should be prevented to maintain consistency. **Fig. 2** illustrated the flow of replication status confirmation.

3.3 Cellular networking for key transmission during large-scale disaster

The final component of our proposed long distance synchronization method is the network transmission control. In addition to wired network connection, we supplemented a wireless network connection to provide connectivity between main and backup server.

This cellular networking components is added to provide additional safety net for transmitting data to the backup server in the event of large scale disaster which causes wired connection to be unavailable. Higher priority is assigned to the key-only packet information. By utilizing the additional cellular connection from main to backup server, we transmit at least the key and transaction identifier to the backup server. Due to its small value size, the key and transaction ID packet can be transmitted quickly.

If the wireless connection is also unavailable, we store the key-value pairs into permanent storage. After the connectivity is restored, we compare the key-value pairs in main server to the key-value pairs in backup server and resolve any conflicts by transaction identifier matching.

4. Evaluation

To evaluate the effectiveness of our proposed method where key is separated from its corresponding value, we compared its latency performance to traditional approach where key is transmitted together with its value. We developed two prototypes of KVS, one for our proposed key-value separated method and one for traditional approach where key is not separated from its value. We are focusing our evaluation on the software processing technique to achieve consistency protection..

4.1 Experimental setup

We evaluated the performance of our proposed method by measuring the time required to achieve consistency protection and prevent conflicts. We start the measurement after request is received by main server, when consistency protection is achieved, and when synchronization finishes. In our proposed method where key is separated from its value, this corresponds to the time where the key information reaches the backup server. In traditional approach, consistency protection corresponds to the time when the replication is finished. **Fig. 3** and **Fig. 4** illustrates the difference in a sequence chart. In our proposed key-value separation method, consistency protection time and synchronization finish time is two distinct measurement points. In traditional no separation method, they are the same point.

In our evaluation, we focused our evaluation on the latency, not the throughput. This decision is based on our assumption

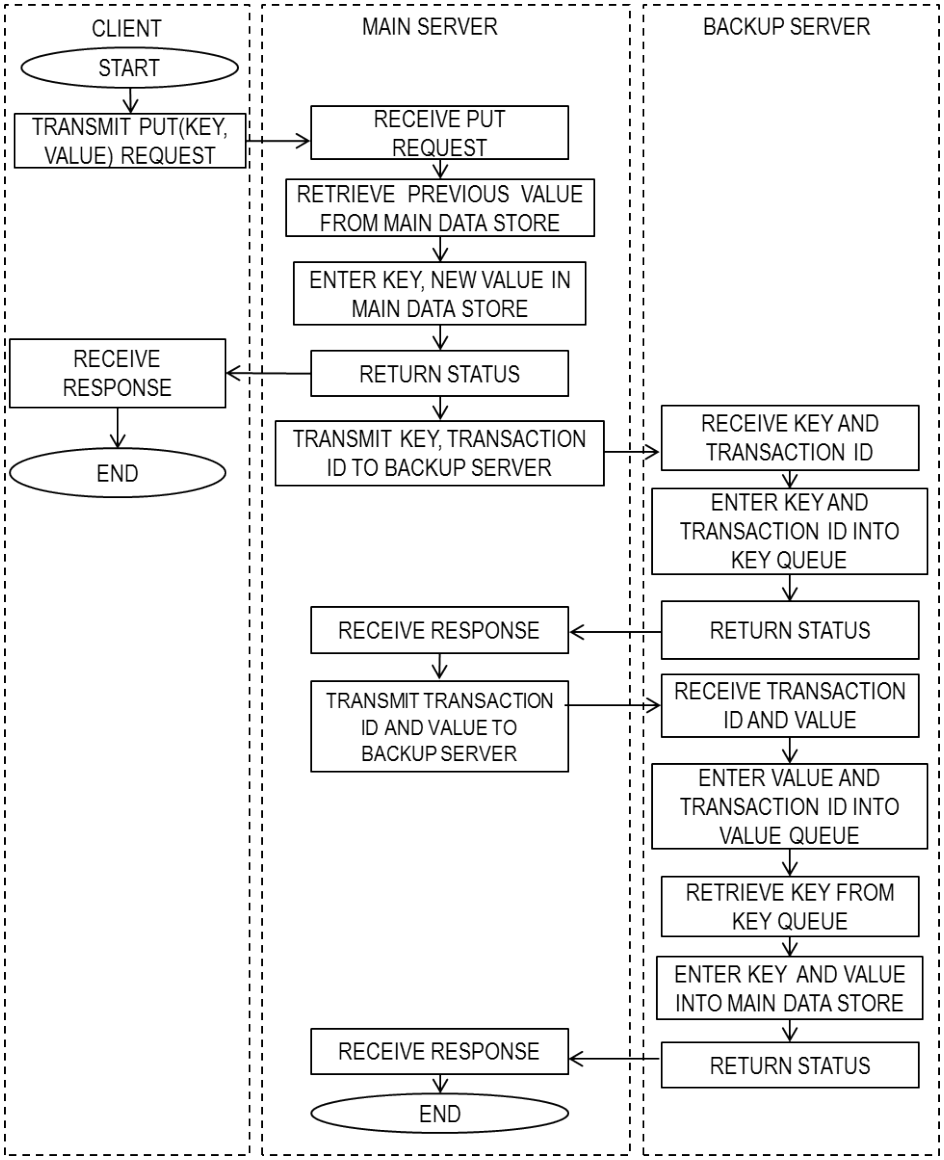

Fig. 1 Flowchart of replication from a put request by local client until the end of replication

Table 1 Evaluation environment

Parameter		Value
Hardware	# of Host	2xQuantaPlex
	CPU	Intel Xeon® E5-2640v3 (Haswell) x 2
	Memory	DDR4-1866 64GB
	NIC	NIC Intel X540-AT2 10GbE
Software	OS	CentOS 7 x86_64
	Java	Java SE 8u77

that the backend storage such as our KVS is not limited by the database response to application, but by the application response to the client. Significant processing is performed by the application, while storing the data to the in-memory KVS is finished in comparatively shorter time.

We performed the evaluation on bare metal servers with specifications as listed in **Table 1**. The servers are connected via IPv4 network as illustrated in **Fig. 5**.

4.2 Evaluation results and discussion

Fig. 6 depicts the comparison between our proposed key-value separation method and traditional no separation method for the required time to achieve consistency protection, for simulated network latency of 5 ms, and 40 ms respectively.

With small-sized value of less than 10 kB, our proposed method provided no advantage compared to traditional approach, the time to achieve consistency protection is approximately the same. However, when the value-size grows bigger, our proposed method provided faster time to achieve consistency protection. In traditional approach, the time it takes to achieve consistency protection grows with value size, which results in longer time to achieve consistency protection. In our proposed method, consistency protection is achieved when the key reaches backup site. Therefore, time to achieve consistency protection in our proposed method does not grow with value size.

Time to achieve consistency protection grows proportionally to the simulated latency, because latency corresponds to the required time to finish round-trip communication from main to backup server.

Fig. 7 depicts the total synchronization time for our proposed key-value separation method and traditional no-separation

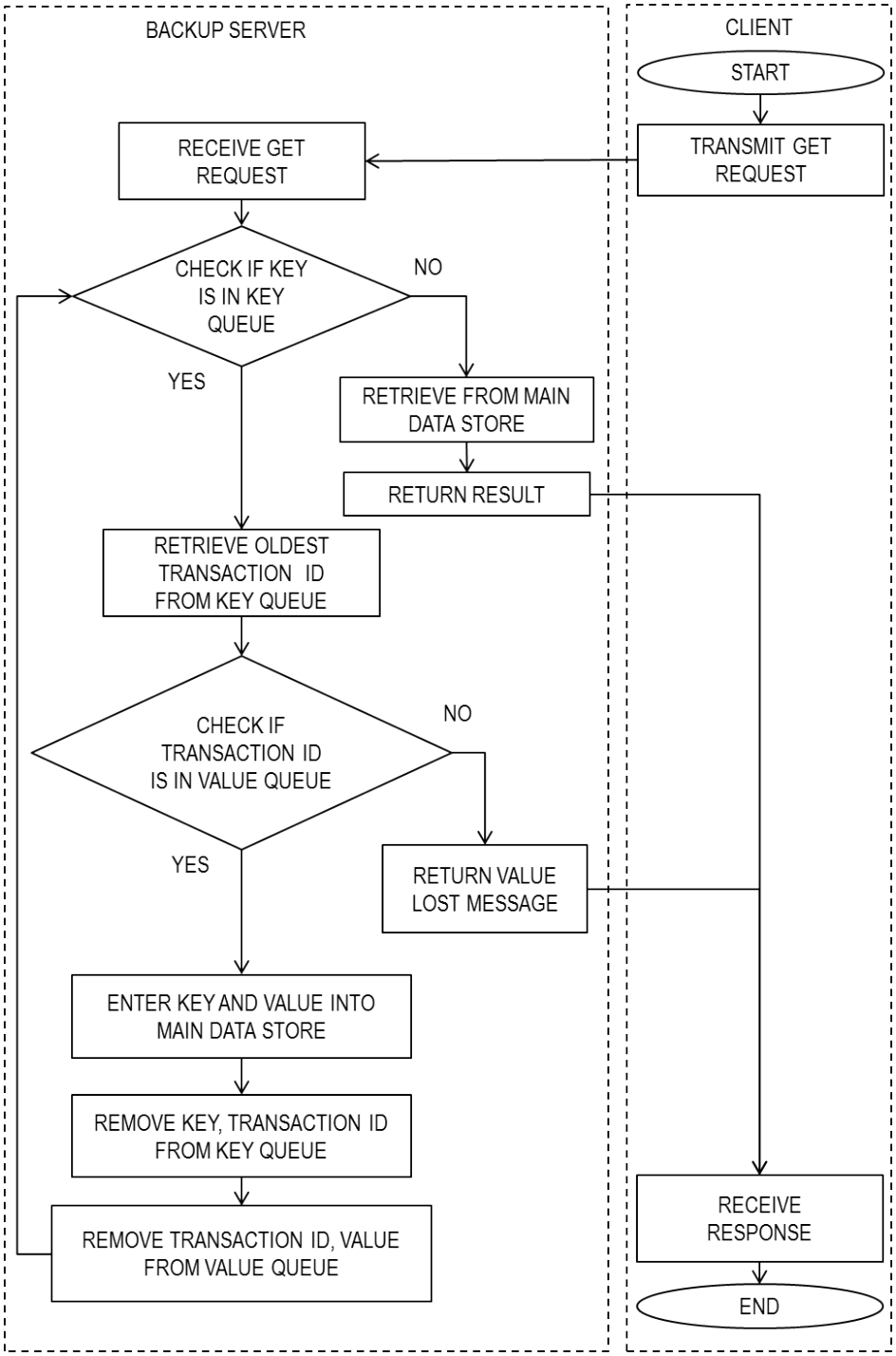

Fig. 2 Flowchart of replication from a put request by local client until the end of replication

method. As the tradeoff for faster consistency protection time, the total time for synchronization is longer. The reason for this behavior is our proposed method requires two network transmissions for finishing synchronization. On a side note, we believe parallel processing can improve the performance of our proposed method to alleviate this performance tradeoff.

Similar to the time to achieve consistency evaluation, time to finish synchronization grows proportionally to simulated latency. Due to the multiple round-trip communication required to finish the synchronization, our proposed method requires longer to finish the synchronization.

Finally, **Fig. 8** depicts the additional consistency protection our proposed method provides for cases where our KVS is operating at maximum throughput. In this evaluation, at the completion moment of the 1000th transaction, we simulated disaster and evaluate how much of the original 1000 transactions can be protected by our proposed key-value separation method and traditional no separation method. We utilized the latency value of 5 ms. We can observe that, consistent with our previous result, our proposed method provides better consistency protection at bigger value size.

In small value size of less than 10 kB our method performed

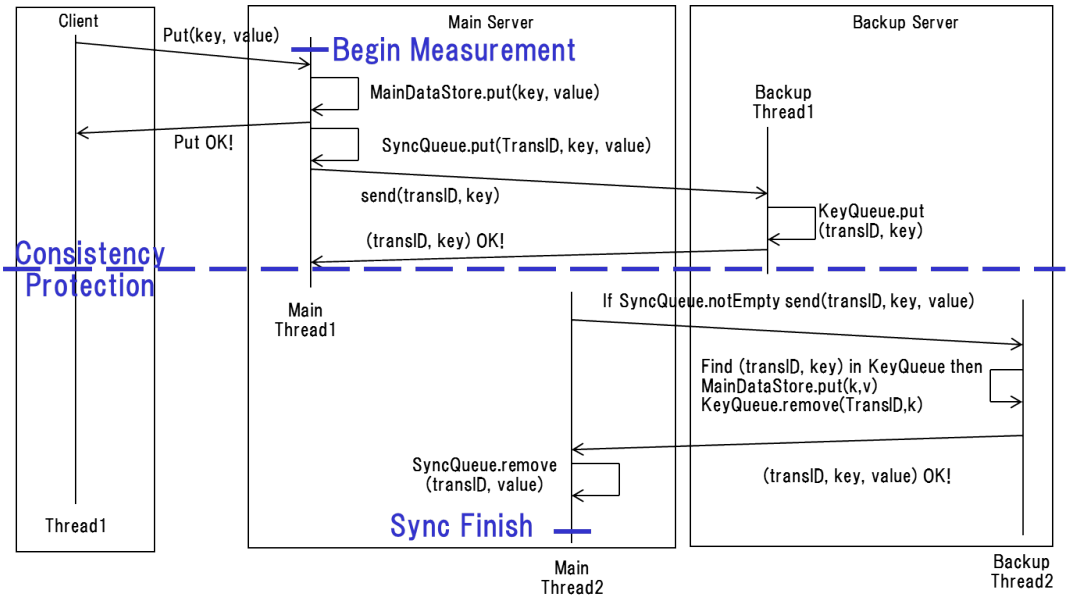

Fig. 3 Measurement point for our proposed method where key is separated from its value

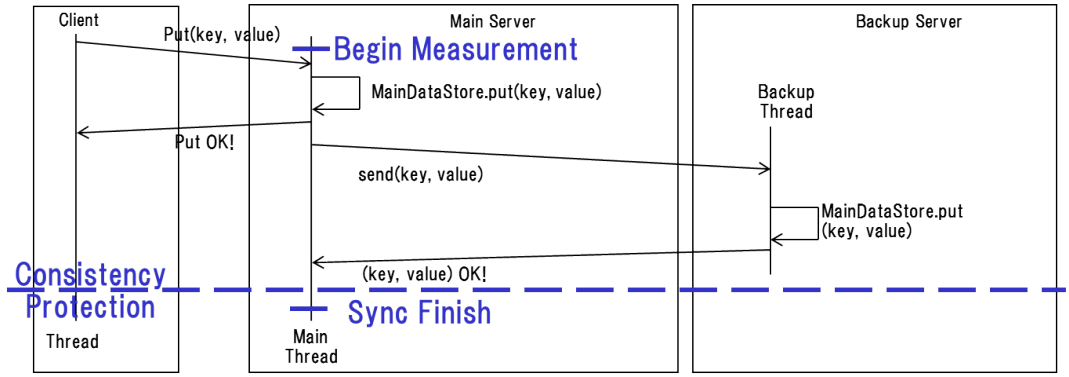

Fig. 4 Measurement point for traiditional approach where key is transmitted together with its value

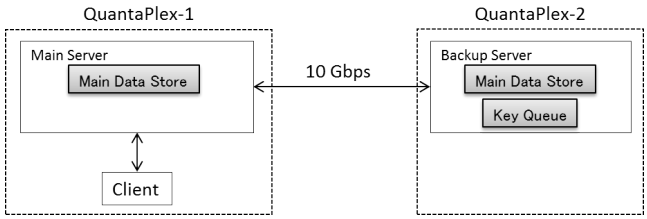

Fig. 5 Experimental setup

worse compared to traditional approach. Although, the number of inconsistencies is similar, due to our design of prioritizing transmission of key information, there are some data whose key exist without its value.

However, for value bigger than 10 kB our method provided additional consistency protection compared to traditional approach. Since our method does not have the bottleneck of value size, the number of inconsistencies is constant and proportional to the latency, while in traditional approach, the number of inconsistencies grows with value size.

For value size of 10 kB, 100 kB, 1 MB, and 10 MB, the amount of inconsistencies reduced by our proposed method corresponds to 10%, 39%, 64%, and 90% respectively. Nevertheless, we believe our approach has significant advantage for processing large data size compared to traditional approach of sending key-value simultaneously.

5. Related research

Existing KVS performs replication without separation between key and its value or unable to support replication at all. Memcached[7], which is one of the first in-memory KVS, does not support cross datacenter replication. Other popular in-memory KVS such as Apache Cassandra[8], Apache Hbase[9], and Pivotal Gemfire[10][11] perform replication by transmitting the key and value together, without separation. Therefore, if the value size grows, for example in the case of unstructured data processing, our proposed method will provide better consistency protection. It is worth to point out these in-memory KVS does not support strong consistency model by default, therefore some risk of inconsistencies between main and backup server may be acceptable for their eventual consistency[1] programming model.

Alternative in-memory KVS which support strong consistency models exist, such as Basho Riak[12] and Apache Kudu[13], but are less well-known. These in-memory KVS is more suitable to our proposed method due to the requirement of maintaining strong consistency models. However, both Riak and Kudu have

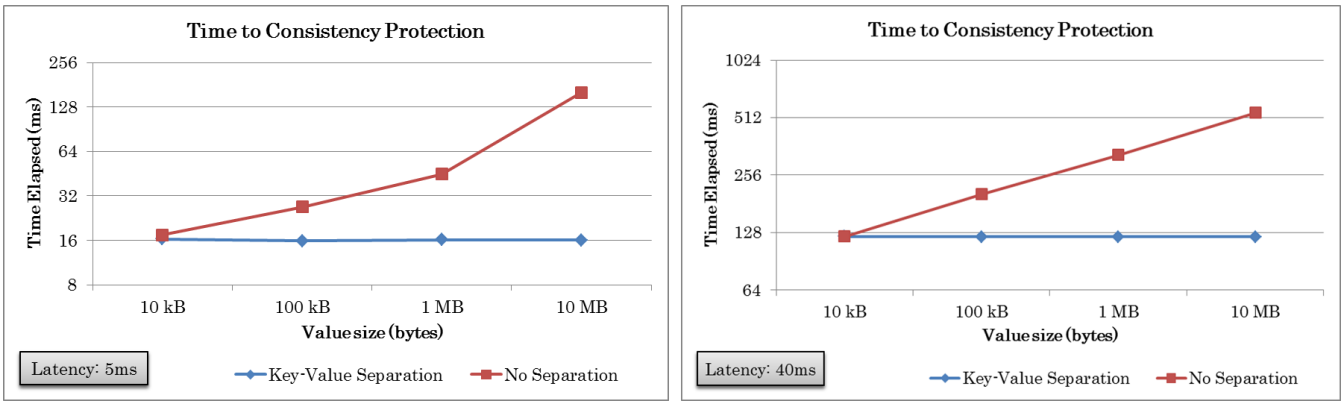

Fig. 6 Time to achieve consistency protection (lower is better)

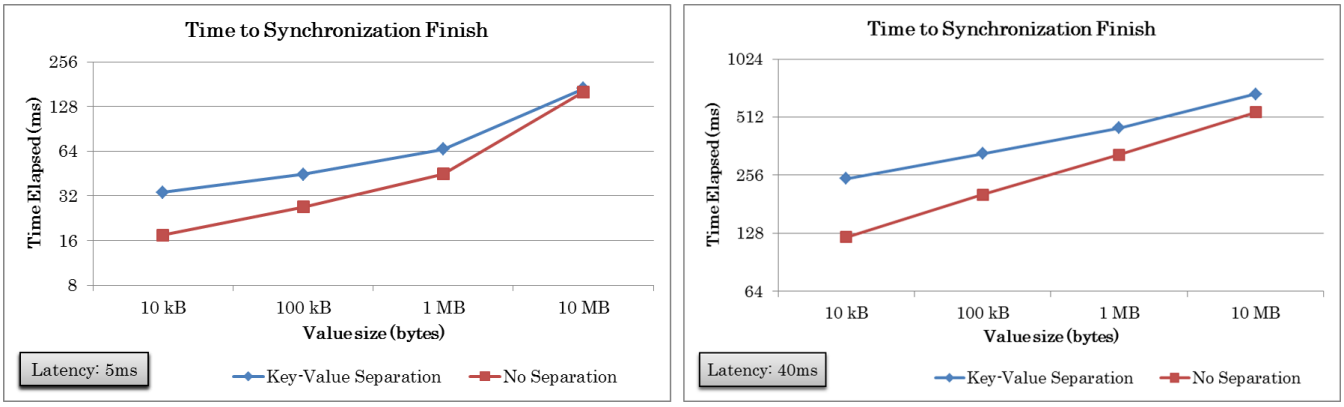

Fig. 7 Time to finish synchronization (lower is better)

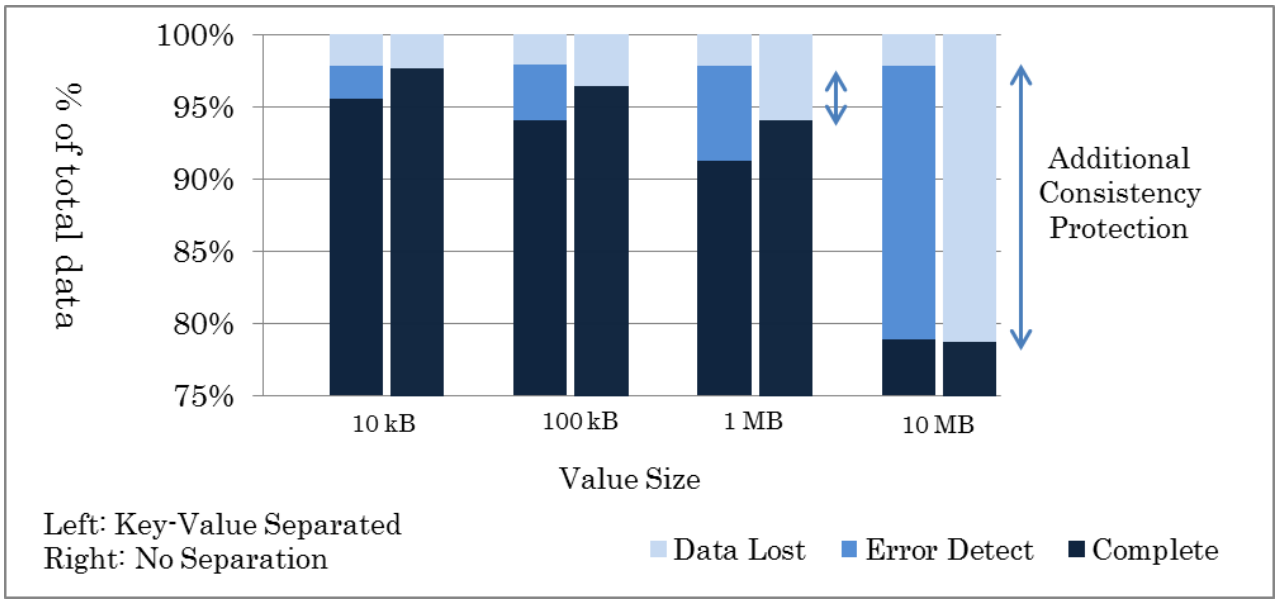

Fig. 8 Additional consistency protection obtained from key-value separation

not support multi datacenter replication yet.

Other techniques [14][15], which are not limited to in-memory KVS, involving parallel processing for accelerating long distance replication, has been proposed as well. However, both techniques did not split the identifier from the main data, and requires the key-value replication to be completed for preserving consistency.

In the case of prioritizing identifier before the main data, [16] transmitted metadata before the actual data for provisioning purpose. However, it is not targeted at system with high transaction write workload, thus lacking transaction ordering by transaction number and the status confirmation check mechanism to ensure that the data currently being read is the most recent copy or not. Therefore, consistency protection is not a focus of the technology, which causes inconsistent condition in the case of high-write workload.

6. Conclusion

We proposed a synchronization method for KVS to improve consistency protection in long distance data synchronization. Our method separate key from its value and prioritizes the key for transmission to the backup location, and utilizes this key information and transaction ordering mechanism to reduce the number of inconsistencies caused by large-scale disasters. By experimentation on synchronization workload, we demonstrated that our method provided 39% reduction of inconsistencies for 100 kB value size, compared to traditional no separation approach. From these results we believe our method could provide better consistency protection for processing of big data.

There are remaining improvements which can be made. Our method made the tradeoff of faster consistency protection for longer synchronization time. However, it is possible to reduce the time to finish synchronization in our proposed method to an amount comparable to traditional method by parallel processing for transmitting the key and value data simultaneously. However, such implementation will require improvement on replication flow control to handle the additional complexity of state management.

Additionally, we developed our method towards in-memory KVS application. Filesystem works in similar capacity, and our method may be of use. However, applying our method to filesystem may require further investigation for filesystem-specific processing, such as how to deal with identifier-only information when interacting with caching system.

References

[1] Sakr, S., Gaber, M. (Eds.): *Large Scale and Big Data: Processing and Management* CRC Press (2014).
[2] Lamport, L.: Paxos made simple, *ACM Sigact News*, Vol.32, No.4, pp.18–25 (2001).
[3] Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm, *2014 USENIX Annual Technical Conference (Usenix ATC 14)*, pp.305–314 (2014).
[4] Zavoral, F., Yaghob, J., Pichappan, P., El-Qawasmeh, E. (Eds.): *Networked Digital Technologies, Part II*, Springer (2010).
[5] Hoffmann, J.: *Mixed mode synchronous and asynchronous replication system*, U.S. Patent 8 301 593, December 17, 2009.
[6] Arnon, D.: *Write pacing*, U.S. Patent 7 702 871, April 20, 2010.
[7] Memcached (online),
available from ⟨https://memcached.org/⟩ (accessed 2016-8-4).
[8] Lakshman, A., Prashant, M.: Cassandra - A Decentralized Structured Storage System, *ACM SIGOPS Operating Systems Review*, Vol.44, No.2, pp.35–40 (2010).
[9] Apache Hbase (online),
available from ⟨https://hbase.apache.org/⟩ (accessed 2016-8-4).
[10] Williams, J. W., Aggour K. S., Interrante J., McHugh J., Pool E.: Bridging high velocity and high volume industrial big data through distributed in-memory storage & analytics, *Big Data (Big Data), 2014 IEEE International Conference on*, pp.932–941, IEEE (2014).
[11] Pivotal Gemfire Multi-site (WAN) Configuration (online),
available from ⟨http://gemfire.docs.pivotal.io/docs-gemfire/topologies_and_comm/multi_site_configuration/chapter_overview.html⟩ (accessed 2016-8-4).
[12] Basho Riak Replication Model (online),
available from ⟨http://docs.basho.com/riak/latest/ops/advanced/strong-consistency/⟩ (accessed 2016-8-4).
[13] Cloudera Kudu (online),
available from ⟨http://www.cloudera.com/documentation/betas/kudu/0-5-0/topics/kudu.html⟩ (accessed 2016-8-4).
[14] Ramakrishnan, K. K.: *Pipelined Data Replication for Disaster Recovery*, U.S. Patent Application 20140040206, February 6, 2014.
[15] Zheng, W.: *Virtual Machine-based On-Demand Parallel Disaster Recovery and the Method Thereof*, U.S. Patent 8 161 321, April 17, 2012.
[16] Niki, Y.: *Information processing system and method of controlling the same*, U.S. Patent 8 688 632, April 1, 2014.