Improved Resilience through Extended KVS-Based Messaging System

Masafumi KINOSHITA^{†,††a)}, Osamu TAKADA[†], Members, Izumi MIZUTANI[†], Takafumi KOIKE^{†††}, Nonmembers, Kenji LEIBNITZ^{††††}, Member, and Masayuki MURATA^{††}, Fellow

SUMMARY In the big data era, messaging systems are required to process large volumes of message traffic with high scalability and availability. However, conventional systems have two issues regarding availability. The first issue is that failover processing itself has a risk of failure. The second issue is to find a trade-off between consistency and availability. We propose a resilient messaging system based on a distributed in-memory key-value store (KVS). Its servers are interconnected with each other and messages are distributed to multiple servers in normal processing state. This architecture can continue messaging services wherever in the messaging system server/process failures occur without using failover processing. Furthermore, we propose two methods for improved resilience: the round-robin method with a slowdown KVS exclusion and the two logical KVS counter-rotating rings to provide short-term-availability in the messaging system. Evaluation results demonstrate that the proposed system can continue service without failover processing. Compared with the conventional method, our proposed distribution method reduced 92% of error responses to clients caused by server failures.

key words: KVS (key-value store), in-memory data grid, messaging queue, distribution system

1. Introduction

In the big data era, Internet of Things (IoT) and Machine-to-Machine (M2M) systems are expected to include 26 billion of connected devices [1] and these devices will send large volumes of message traffic. Messaging systems are required to process this huge traffic at a high throughput, scalability, and availability, which are the most important features of mission-critical systems.

In our previous work, we proposed a high-throughput messaging system based on a distributed in-memory *key-value store* (KVS) to process large volumes of messages and to achieve high availability [2]–[4]. The proposed KVS-based messaging system stores duplicated messages in an in-memory queue structure. However, there still exist two

issues regarding availability.

The first issue is that failover processing itself has a risk of failure. Generally, mission-critical systems implement shared data and failover processing for providing *high availability* (HA) services [5]. Failover processing includes application restart, process initialization, and recovery of data. These processes consist of special application-dependent processes, as well as common processes, such as health check and error detection of hardware/software. However, in recent years, catastrophic service failures of mission-critical systems with failover processing [5]–[7] have frequently been reported. Causes of these service failures are usually software or hardware defects, and it is very difficult to exhaustively identify these defects at the system testing stage because all cases of failover processing, e.g., complex problems caused by only theoretically occurring defects, can hardly be tested. Therefore, a highly available messaging system without failover processing is needed.

The second issue is to balance between consistency and availability. For messaging systems, a strong consistency of messages and message queues is the highest requisite to maintain reliable messaging and billing processing. They also require maintaining the state of the messaging process and the internal queue lock (these functions are denoted as *queue transactions*). Consistency is a common issue in general KVSes and distributed computing [11]. According to *Consistency Availability Partition tolerance* (CAP) terminology [8], in the KVS we can trade off between consistency and availability.

Here, we should have an approach that the consistency in the messaging system is guaranteed by the KVS functions, and the availability is improved by our proposal in this paper. More specifically about the availability of a messaging system, not only 365 days non-stop service is mandatory as *long-term-availability*, but also *short-term-availability* is required, e.g., even during a transient state while a failed server is being isolated or a traffic congestion is being eliminated, that the messaging service is provided continuously without a performance degradation. In this paper, we propose a fabric messaging system without failover processing. *Fabric messaging* denotes the distribution of messages to multiple servers in normal processing state to avoid failover processing. This system has the following two features and advantages.

- The messaging system architecture based on dis-

Manuscript received May 30, 2014.

Manuscript revised October 7, 2014.

Manuscript publicized December 11, 2014.

[†]The authors are with the Yokohama Research Laboratory, Hitachi Ltd., Yokohama-shi, 244–0817 Japan.

^{††}The authors are with the Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan.

^{†††}The author is with the Telecommunications & Network Systems Division, Hitachi Ltd., Kawasaki-shi, 212–8567 Japan.

^{††††}The author is with the Center of Information and Neural Networks (CiNet), National Institute of Information and Communications Technology, and Osaka University, Suita-shi, 565–0871 Japan.

a) E-mail: masafumi.kinoshita.rt@hitachi.com

DOI: 10.1587/transinf.2014NTP0015

tributed in-memory KVS can provide *long-term-availability*, i.e., continuing its service wherever in the messaging system server/process failures occur, by distributing messages to multiple servers, as well as it can guarantee strong consistency of the messages and message queues by using KVS functions and the Paxos [9], [10] protocol.

- The distribution method of messages to servers by using round-robin with a slowdown KVS exclusion and two logical counter-rotating KVS rings can achieve *short-term-availability* even during an underlying network failure and/or slowdown of servers.

The rest of this paper is organized as follows. The background and issues of messaging systems are introduced in Sect. 2. Section 3 presents the system architecture and design. Section 4 shows the implementation and the performance evaluation results on availability. Section 5 describes related work and Sect. 6 concludes this paper.

2. Background and Issues

2.1 Outline of the Messaging System

Figure 1 outlines an example of the system structure in messaging services. Messaging systems are widely used for a large variety of services such as e-mail service, collection of sensor data, e.g. smart meters, mobile health, or transportation information. Main function of messaging systems is to *store-and-forward* messages from source clients such as mobile devices to the final destination or to the next-hop messaging system.

First, the messaging server receives the message (1) and stores it in a message queue on its disks or in the KVS while waiting for its transmission to the destination (2). After storing the message, it responds to the client immediately. If the message cannot be stored for any reason, e.g., due to KVS overflow, an error response is sent to the source client (3). The received message is sent to the final destination or to the next-hop messaging system (4). The received message is sent instantaneously, but may also be delayed if the destination or the next-hop messaging system is temporarily unavailable. The messaging servers keep retrying the delivery for a certain period of time, for example, several hours or a few days. Finally, the server deletes the message if the message was successfully sent or if a retransmission timeout occurred.

2.2 Risk of Failover Processing

Figure 2 outlines an example of failover processing. Mission-critical systems usually have HA clusters for continuous service when their components fail. HA clusters detect hardware/software failures and immediately restart the application on another standby system, which is referred to as *failover*.

Conventional messaging systems have a risk of failover

Fig. 1 Messaging system structure.

Fig. 2 Risk of failover processing.

processing. Similarly, our previously proposed system in [2]–[4] also partly has this risk because it uses recovery processing in which another system (messaging server) on standby gets all messages stored before the failure in KVS.

To solve the issue caused by failover processing in the messaging system, we take advantage of distributed in-memory KVS. Generally, distributed in-memory KVS is used for high-throughput and scalability. However, we use it for improving availability of the messaging system in this paper. In order to remove failover processing, we follow the approach of *fabric messaging* that distributes messages to all servers during a normal processing state. A “fabric” is a topology in which nodes pass data to each other through interconnecting nodes in a mesh fashion. In data center network research, switch fabrics are well known [15], [16]. In this paper, we propose a fabric architecture on the application layer containing the data store for solving the above-mentioned failover processing issue of messaging systems (see Sect. 3.1).

2.3 Trade-off between Consistency and Availability

2.3.1 Message Queues on a Distributed KVS Ring

For scalability of the data store, a general distributed KVS distributes data (key, value) by *consistent hashing* [12], [13], and a cluster of distributed KVS is configured by using *range partitioning* [12]–[14] (the cluster of distributed KVS is denoted as *KVS ring*). In the KVS ring, each server (*coordinator* in [12], [13]) is responsible for the region between itself and the previous server on the ring.

Fig. 3 Applying conventional method in real system.

Our previously proposed messaging system [2]–[4] simply applied KVS technology, therefore, the consistency of messages cannot be maintained when split-brain occurs as shown in Fig. 2. To maintain a strong consistency, we use Paxos [9], [10], a protocol for obtaining consensus in interconnected unreliable processors, which is widely used in many distributed processing systems [11], [14].

However, even if both general distributed KVS technology and Paxos were simply applied to the messaging system as shown in Fig. 3, there would be new problems that are described in 2.3.2.

In Fig. 3, each KVS is assigned queues based on range partitioning. Each queue is stored in three KVSes, and can be in either *master*[†] or *non-master* state. The *master* queue is responsible for queue transactions, such as en-queuing or dequeuing of messages, and for message replication of the two *non-master* queues. If a KVS failure is detected in a KVS ring by Paxos, the faulty KVS is isolated from the ring and one of the *non-master* queues becomes the new *master* queue as alternative to the previous *master* queue on the faulty KVS. A messaging server selects a KVS with a *master* queue by using consistent hashing and sends messages to the selected KVS.

2.3.2 Two Problems in the Messaging System

Figure 4 outlines the new problems when applying conventional methods to the messaging system. When constructing the messaging system, as shown in Fig. 3, the KVSes are connected to the underlying network, which consists of more than one network device (each device has its own standby device in case of a failure). Considering a route change between switches in case of failure at a single switch or multiple switches, the route stabilization time is about

[†]We use the term *master* as reference in [11] because the proposed system uses internal locks for queue transactions and it. *Master* includes functions of *leader* in Paxos [9], [10], [14].

Fig. 4 Problems of applying conventional methods.

several seconds or more than 10 seconds, respectively.

A temporal performance degradation of a server is another example. As shown in Fig. 4, when the server of KVS-C fails, the two servers' performance of KVS-A and KVS-B having the responsibility for non-master queues, degrades due to multiple reply-timeouts of KVS-C until the detection and isolation of KVS-C's failure; Typically, more than 10 seconds is needed for a server failure detection, (see Fig. 3). Furthermore, a heavy workload background job also degrades server performance temporally. Those are denoted as *slowdown* of a server. Above-mentioned examples may lead to the following two problems of the *short-term-availability* in the messaging system as shown in Fig. 4.

(1) Large values of KVS failure detection timer

To avoid false detections of a server-failure during the route stabilization of the underlying network, the value of the detection time for the KVS failure must be set to larger than the route stabilization time of the network, i.e., several to more than 10 seconds. Consequently, all messaging servers must wait for the response from the faulty KVS until the KVS failure detection timer expires. The same problem happens, when a slowdown occurs. One example of this adverse effect is that the messaging server cannot reply to the mobile devices (source clients) for over ten seconds, while wasting wireless resources and degrading messaging service quality (see Fig. 4 (1)).

(2) Concentration of message queue load after KVS failures

When KVS-C in Fig. 4 fails, KVS-D becomes the new master after the KVS failure detection time, and the non-master queue is designated as the new master queue described in Sect. 2.3.1. The designation order of the new master queue depends on the KVS ring's direction. For example, if both KVS-C and KVS-D fail, KVS-E is designated as the new master, therefore, it has to process three master queues of all three KVSes (C+D+E). In this situation, the load of

message traffic concentrates on KVS-E, which can lead to performance degradation (see Fig. 4 (2)).

The first problem described above is due to the fact that the detection time for the KVS failure on the underlying network or a server slowdown can be relatively long compared to the messaging service itself (tens of milliseconds for messaging versus more than ten seconds for network stabilization or server slowdown). The second problem arises from the nature of the distributed KVS since it is critical for mission-critical systems, e.g., carrier grade systems, to continuously provide services when multiple server failures occur.

In Sect. 3.2, we propose distribution methods to solve these problems and provide the *short-term-availability*, while guaranteeing the consistency of the message by the Paxos protocol used in the KVS.

3. Proposed Architecture and Distribution Methods

3.1 Architecture of Fabric Messaging System

The architecture of the proposed fabric messaging system is shown in Fig. 5. Both logical structure/functions and physical configuration are described below.

3.1.1 Logical Structure and Functions

We describe the logical structure and function of proposed fabric messaging system where the following numbers correspond to those in Fig. 5.

- (1) The load balancing module dispatches incoming messages from the source clients to the receiving module in the same way as the conventional system. It monitors the TCP ports of the receiving modules to avoid dispatching to a faulty receiving module.
- (2) Multiple receiving and sending modules are interconnected via multiple KVSes. Both the receiving module and the sending module are stateless and operate cooperatively and independently through the message queues in the KVSes.
- (3) The receiving module selects a KVS by round-robin with a rule that excludes KVSes in a faulty and/or slowdown state, then stores the message at the selected KVS (see Sect. 3.2.1). Therefore, the receiving module can store messages, regardless if there are KVS failures and can continue services.
- (4) The KVS on a server is logically linked to shape a directional ring that includes multiple KVSes and provides distributed KVS. Message queues are deployed on the KVS ring as mentioned in Sect. 2.3.1. Messages and message queues are handled with the Paxos protocol as distributed KVS consisting of three KVSes. Each KVS has multiple message queues. In Fig. 5, there are three queues, the topmost one is a master queue and the lower two are non-master queues. The

Fig. 5 Fabric messaging system architecture.

receiving module en-queues and the sending module de-queues messages via the master queue.

Functions for high availability such as KVS failure detection, isolation of the faulty KVS, and the master/non-master KVS reassignment, are based on a basic distributed KVS described in Sect. 2.3. KVSes can continue service such as en-queuing and de-queuing messages after failure detection regardless of which KVS has a failure (regarding availability within the failure detection, see Sect. 3.2.1).

- (5) Multiple sending modules get messages from multiple KVSes and send them to the destination. Therefore, there is enough redundancy for the messaging service even if failure/slowdown of the sending module occurs. In detail, the sending module gets a message from one of the master message queues and sends it to the destination. If the message is successfully received by the destination, the sending module removes the message from the master messaging queue. The sending module sets an internal lock to the messaging queue while accessing it to arbitrate access conflicts. The sending module preferentially gets messages from a local KVS, i.e., located on the same physical server, than from non-local KVSes to reduce processing overhead.

3.1.2 Physical Configuration and Features for High Scalability and Availability

The fabric messaging system consists of N units of load balancers and servers, as well as network devices (not shown explicitly in Fig. 5). All the servers have a homogeneous configuration where the receiving module, a distributed KVS, and the sending module are all located in one server. This configuration makes it easy to add/delete servers in this system.

KVSes use Paxos for communication within their logical KVS ring for maintaining strong consistency, even in the case of network faults or split brain. We consider two KVS rings that are independent of each other in our architecture. If a server failure occurs in one KVS ring or its modules,

this system can continue with the messaging service by using the other KVS ring. Both KVS rings are connected to different networks and therefore, this system can continue services even when a network becomes disconnected.

Based on the proposed fabric architecture, high scalability and the *long-term-availability* of the messaging system can be realized.

Note that regarding messages from a specific source to destination, the message delivery is guaranteed as long as the destination is ready to receive the message, but the order of message delivery is not necessarily guaranteed because there exist multiple paths (KVSes) between the receiving and sending module. Adding a KVS selection condition could prevent message reordering, e.g., a pair of source and destination client addresses is mapped to one specific KVS.

3.2 Distribution Methods for Improving Short-Term-Availability

3.2.1 A Round-Robin Method with Slowdown KVS Exclusion

To solve the problem described in Sect. 2.3.2 (1), i.e., when the detection time for the KVS failure on the underlying network or a server slowdown is three orders of magnitude longer than the messaging service itself, we introduce the KVS status as *slowdown* or *no-slowdown*. A receiving module selects a KVS by the round-robin method with a slowdown KVS exclusion, instead of the consistent hashing method basically used in the conventional KVS.

In detail, the receiving module monitors the elapsed time that starts at the time of transmitting messages to a KVS until reception from the KVS. The receiving module has a threshold of the elapsed time for each KVS (this threshold is denoted as *slowdown detection time*). If the elapsed time exceeds the *slowdown detection time*, the receiving module determines the KVS state as *slowdown*. When the receiving module determines the KVS state as *slowdown*, it avoids storing messages in that KVS and stores them in another KVS in a *non-slowdown* state as shown in Fig. 6.

By monitoring each KVS with a *slowdown detection time*, its value set to several hundred milliseconds, we can avoid the longer detection time for the KVS failure/server-slowdown. An optimal value of the *slowdown detection time* is evaluated in Sect. 4.3.

3.2.2 Two KVS Counter-Rotating Rings

In order to solve the problem “concentration of message queue load after KVS failures” described in Sect. 2.3.2, we propose the message distribution method with two counter-rotating KVS rings as shown in Fig. 7. The KVS has three queues, the leftmost is the master queue and the other two are non-master queues. Both KVS rings have opposite directions of processing order.

Fig. 6 Distribution method avoiding slowdown KVS.

Fig. 7 Distribution method with 2 KVS rings.

In normal state a receiving module distributes messages to the master queues by round-robin between both KVS rings (see Fig. 7(1)). The master queues make the message replication for the two non-master queue. If a KVS failure/slowdown happens, it would impact the two KVSes that have the master queue sending replicated messages to the non-master queues of the faulty KVS until the faulty KVS is isolated (see Fig. 7(2)). For example, a slowdown of KVS1-C in Fig. 7 influences KVS1-A and KVS1-B. At that time, if a receiving module can determine KVS1-B slowdown (described in Sect. 3.2.1), it skips with the next message to KVS2-B in the other KVS ring, which KVS1-C failure does not impact.

If a server failure happens, a receiving module can also determine the KVS slowdown and skips with the next message to the normal KVS. After that KVSes detect the faulty KVS and change one of the non-master queue of KVS to the master queue. For example in Fig. 7, if a failure of server-C happens, the non-master queues of KVS1-D and KVS2-B become master queues (see Fig. 7(3)). This divides the load onto two servers and is more effective when multiple server failures occur simultaneously, e.g., failures

of server-C and server-D (see Fig. 7 (4)). With the conventional method that has only a single KVS ring, KVS-E has to process the data of three KVSes (C+D+E) when KVS-C and KVS-D fail. On the other hand, with our proposed method, KVS1-E and KVS2-B only need to process data of two KVSes under the same situation. Thus, the proposed distribution method reduces the negative impacts on the service caused by server/KVS process failures.

4. Implementation and Evaluation

4.1 Implementation and Methodology for Evaluation

Receiving and sending modules were implemented based on an event-driven architecture [17] developed in the C language. We implemented KVSes, which have a key-value data structure, in Java[†] and added functions for queue transactions to the KVSes. The messaging system for the evaluation consists of 5 receiving modules, 5 sending modules, and 10 KVSes. There are 2 logical KVS rings, each consisting of 5 KVSes. Each KVS has 18 GB of memory for storing more than a million messages.

We evaluate the *short-term-availability* provided by proposed two methods described in Sect. 3.2 and the *long-term-availability* of the fabric messaging system described in Sect. 3.1. For the evaluation of the proposed messaging system, we assume an e-mail system with large volume of message traffic. From our experience in commercial e-mail systems, we choose an e-mail size of 30 KB. A test client program generates the workload using the Simple Mail Transfer Protocol (SMTP) to the receiving modules and the messaging system forwards them to a test destination server.

4.2 Verification of Detection of Slowdowns

In order to verify the effect of the round-robin method with a slowdown KVS exclusion described in Sect. 3.2.1, we compared the throughput of two messaging systems, one is with the proposed round-robin method with a slowdown KVS exclusion, another is with conventional consistent hashing method. Figure 8 outlines the test environment of the evaluation. We let server-D fail while processing the workload and monitored the throughput and the error responses replied to the test client program from all the receiving modules. The test client program transmitted the workload to receiving modules at a rate of 1200 msg/sec that can be processed stably under one server failure in this evaluation environment.

Throughputs of the two messaging systems are shown in Fig. 9. For the proposed method, the throughput remains stable at the time and after the server failure. In contrast, for the conventional consistent hashing method, the throughput is decreased temporally about 15 seconds after

Fig. 8 Method of evaluation of messaging system.

Fig. 9 Throughput of messaging systems.

Fig. 10 Number of error responses.

the server failure. The number of error responses is shown in Fig. 10. Compared to the conventional method (2379 error responses), the proposed method (214 error responses) is decreased by 92%. Thus, it is shown that the proposed method increases the *short-term-availability* of the messaging system.

4.3 Determining the Optimal Slowdown Detection Time

To find an optimum value of the *slowdown detection time* described in Sect. 3.2.1, we evaluated the performance of the proposed system for different values. We use the same

[†]Java is a trademark or registered trademark of Oracle, Inc. in the US and other countries.

Fig. 11 Average and variance of throughput for different slowdown detection time.

Fig. 12 Throughput for different slowdown detection time values (0.1 and 0.4 sec) over time.

test environment as shown in Fig. 8 and evaluated the average and variance of the throughput for different *slowdown detection time* from 0.1 to 1 second.

The average and variance of the throughput are shown in Fig. 11. The average throughput increases in the range from 0.1 to 0.4 seconds for the *slowdown detection time* values, and flattens in the range larger than 0.4 seconds. On the other hand, the variance of throughput decreases in the range from 0.1 to 0.4 seconds. Figure 12 shows the behavior of the throughput for two *slowdown detection time* values, 0.1 and 0.4 seconds, before and after a server failure. The throughput for 0.1 seconds has a high fluctuation, while it is stable for 0.4 seconds. The average throughput for 0.1 seconds is 9% less than the throughput for 0.4 seconds.

In general, it is better to set smaller values for *slowdown detection*, because larger values impact the waiting time of the source clients (mobile devices) as described in Sect. 2.3.2. From the result in Fig. 11, an optimum value of *slowdown detection time* is 0.4 sec.

The reason why the throughput is not stable for 0.1 seconds is an effect of the copying garbage collection of Java. Actually, copying garbage collection happened every second in the test and the process of KVS stopped operation when the detection time value is in the range from 0.1 to 0.3 seconds.

In conventional systems, the duration time of copying garbage collection is negligible. According to previous research on garbage collection of Java [19], the duration time

Fig. 13 Performance evaluation of the proposed messaging system.

for copying garbage collection depends on the memory size and becomes non-negligible when the memory size is larger than 1 GB. We estimate that the duration time of copying garbage collection becomes longer, because each KVS has 18 GB memory and has to store a lot of key-value data including the metadata to achieve queue transactions.

If a KVS halts due to copying garbage collection for more than slowdown detection time, the receiving modules stop transmitting messages to this KVS. As a result, the KVS has nothing to process, leading to a decrease in throughput of the whole messaging system. In addition, we presume this effect of copying garbage collection to be a common problem of KVS-based systems, because many KVS implementations such as Cassandra [12] or Hbase [20] are implemented in Java, and the recent distributed systems are equipped with large memory.

4.4 Impact of Server Failures on Availability

We evaluated the performance of the proposed fabric messaging system from the *long-term-availability* point of view. Figure 13 outlines the test environment of the evaluation. We compared the performance behavior, the throughput and the queue length of the message queue of the proposed system and the conventional system. The conventional system has two KVS same-direction rings because that is same as having a single KVS ring with one direction. The queue length of the message queue reflects the variance of load balancing in the whole system. We applied the optimum value of 0.4 seconds as the *slowdown detection time* to the system.

For mission-critical systems, e.g., carrier grade systems, operators expect the system to handle 2 simultaneous server failures occur and they construct the redundancy system for this worst-case scenario. A single server can stably process 300 msg/sec as shown in Sect. 4.2, therefore, the test client program transmitted the workload to the receiving modules at a rate of 900 msg/sec that can be processed stably when 2 server failures occur. After 60 seconds of transmitting the workload, we first let server-D fail, followed by a

Fig. 14 Throughput of messaging system. (conventional method: same direction of both rings)

Fig. 15 Queue lengths of each KVS. (conventional method: same direction of rings)

failure of server-E.

(1) Conventional Messaging System

Throughput of the conventional messaging system is shown in Fig. 14. Throughput is stable when the first server failure happens, however, when the second server fails, it decreases to zero, meaning that the messaging service stops. After 5 seconds of stopping the messaging service, the service is recovered.

The queue length of the message queue in each KVS is shown in Fig. 15. For example, the queue name “Q1-A” shows the queue of KVS belonging to ring 1 and initially located in server-A. When the server-D failure occurred, the lengths of Q1-E and Q2-E increased. We consider that it is caused by excluding KVS-D. When the server-E failure occurred, the lengths of 6 queues (Q1-A, Q2-A, Q1-D, Q2-D, Q1-E, Q2-E) are increased. We attribute this to the problem in load balancing as described in Sect. 2.3.2 (2). Server-A that included KVS1-A, KVS1-B, and sending module-A has to process the queues of 3 KVSes and the failure of server-E impacted the other KVSes during the failure detection time (described in Sect. 3.2.2) leading to a messaging service stop for 5 seconds.

(2) Proposed Messaging System

The throughput of the proposed messaging system having the two KVS counter-rotating rings is shown in Fig. 16. Compared to the conventional system in Fig. 14, throughput in Fig. 16 remains rather stable when the first and second servers fail. Throughput decreases about 20%, which corresponds to the workload of two messaging (master) queues

Fig. 16 Throughput of messaging system. (proposed method: opposite directions of rings)

Fig. 17 Queue lengths of each KVS. (proposed method: opposite directions of rings)

temporally in an out-of-service state out of the initial five messaging (master) queues.

The queue lengths in each KVS are shown in Fig. 17 for the proposed system. Compared to the conventional system in Fig. 15, the lengths of queues in Fig. 17 are only slightly increased. That is caused by the proposed method reducing the impact of load balancing as described in Sect. 3.2.2.

Thus, it is shown that the proposed architecture and two methods can continue the messaging service even if multiple server failures occur. Therefore, it can provide *long-term-availability*.

5. Related Work

We describe related work from three points of view: messaging systems, failover processing, and distribution methods in distributed systems.

Regarding messaging systems, a queuing system based on distributed in-memory KVS was proposed in [18]. Its queuing function deployed in the KVS is similar to the function of our proposed system. However, it focused on the queuing part only and it did not discuss about the availability of the whole system including the messaging process. In addition, our approach of focusing on availability of distributed in-memory KVS is not the common way in previous research.

Previous study in the risks of failover processing have been reported in [7], [21], [22]. To avoid catastrophic

service failures, they proposed management rules, e.g., monitoring system failures and verifying configurations of failover processing, and preparations, e.g., procedures when system failure happens. It was also mentioned in [22] that designing the system for a concentration of message load after failover processing was important.

The shared nothing architecture [23] is similar to ours when distributing messages in a normal processing state. However, the shared nothing system usually doesn't duplicate messages and needs failover processing or recovery processes to continue service. The significant difference between our proposed architecture and the shared nothing architecture is when it is executed. Our proposed system always executes the same process wherever a server failure happens, while application restart and recovery process in the shared nothing architecture are executed only when a server failure happens. Therefore, our proposed architecture can be more available than the shared nothing architecture.

Regarding distribution method of KVS, consistent hashing is the standard distribution method of KVS [12], [13]. Our proposed method is optimized for queuing and high availability messaging service. As far as we know, there have been only few reports on such systems.

6. Conclusion

For the big data era, we propose a fabric messaging system that has the following functions and advantages.

- The messaging system architecture based on distributed in-memory KVS can provide *long-term-availability*, continuing its service wherever in the messaging system server/process failures occur, by distributing messages to multiple servers, as well as it can guarantee strong consistency of the message/message queue by using KVS functions and the Paxos protocol.
- The distribution methods of messages to servers, by using round-robin with a slowdown KVS exclusion and two logical KVS counter-rotating rings, can achieve *short-term-availability* even during an underlying network failure and/or a slowdown of servers.

Evaluation results show that this system can continue service without failover processing. Compared with the conventional method, our proposed distribution methods reduced 92% of user errors caused by server failures. Furthermore, we determined the optimum value of slowdown detection time in our distribution method.

References

- [1] Gartner, <https://www.gartner.com/doc/2625419>
- [2] M. Kinoshita, G. Tsuchida, I. Mizutani, and T. Koike, "High-throughput messaging system based on in-memory KVS for processing large traffic volume of short message," IEICE Trans. Commun. (Japanese Edition), vol.J96-B, no.10, pp.1206–1216, Oct. 2013.
- [3] M. Kinoshita, G. Tsuchida, and T. Koike, "High-throughput mail gateways for mobile E-mail services based on in-memory KVS," 8th International Conference on Wireless and Mobile Communications (ICWMC 2012), pp.146–153, June 2012.
- [4] M. Kinoshita, G. Tsuchida, and T. Koike, "High-throughput messaging system based on in-memory KVS for processing large traffic volume of short message," IECE Society Conference 2011 (B), p.412, Sept. 2011.
- [5] N. Owada, "How Systems Go Down" (in Japanese), NIKKEI BP, pp.94–105, 2009.
- [6] H. Okabe, "Report of NIKKEI COMPUTER," IT Pro, <http://itpro.nikkeibp.co.jp/article/COLUMN/20120824/417984/>, accessed Sept. 18, 2014. (in Japanese)
- [7] Bank of Japan, "BOJ Report and Research Papers," https://www.boj.or.jp/research/brp/ron_2010/data/ron1011a.pdf, accessed Sept. 18, 2014. (in Japanese)
- [8] E.A. Brewer, "Towards robust distributed systems," 19th Annual ACM Symposium on Principles of Distributed Computing, p.7, 2000.
- [9] L. Lamport, "The part-time parliament," ACM TOCS, vol.16, no.2, pp.133–169, 1998.
- [10] L. Lamport, "Paxos made simple," ACM SIGACT News, vol.32, no.4, pp.18–25, 2001.
- [11] M. Burrows, "The Chubby lock service for loosely-coupled distributed systems," 7th Symposium on Operating Systems Design and Implementation, USENIX Association, 2006.
- [12] A. Lakshman and P. Malik, Cassandra – A Decentralized Structured Storage System, Cornell, 2009.
- [13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Voshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," ACM SIGOPS Operating Systems Review, vol.41, no.6, 2007.
- [14] J. Rao, E.J. Shekita, and S. Tata, "Using paxos to build a scalable, consistent, and highly available datastore," Proc. VLDB Endowment 4.4 (2011), pp.243–254, 2011.
- [15] R.N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," ACM SIGCOMM Computer Communication Review, vol.39, no.4, pp.39–50, Oct. 2009.
- [16] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," ACM SIGCOMM Computer Communication Review – SIGCOMM '09, vol.39, no.4, pp.51–62, Oct. 2009.
- [17] M. Welsh, D. Culler, and E. Brewer, "SEDA: An architecture for well-conditioned, scalable Internet services," 18th ACM Symposium on Operating Systems Principles (SOSP '01), pp.230–243, Oct. 2001.
- [18] Y. Wang, H. Chen, B. Wang, J.M. Xu, and H. Lei, "Scalable queuing service based on an in-memory data grid," IEEE 7th International Conference on e-Business Engineering (ICEBE 2010), pp.236–243, Nov. 2010.
- [19] Y. Miyata, M. Obata, T. Ohta, and H. Nishiyama, "Proposal of GC time reduction algorithm for large java object cache," IPSJ Trans. Programming, vol.5, no.3, pp.29–39, Aug. 2012.
- [20] Apache Hbase Project, "Apache Hbase," <http://hbase.apache.org/>, accessed Sept. 18, 2014.
- [21] Information-technology Promotion Agency, Japan, "High reliability lessons for IT systems," <http://www.ipa.go.jp/files/000038843.pdf>, accessed Sept. 18, 2014. (in Japanese)
- [22] A. Egami, "The measures and background of catastrophic service failures," http://e-public.nttdata.co.jp/topics_detail4/contents_type=20&id=653, accessed May 18, 2014. (in Japanese)
- [23] M. Stonebraker, "The case for shared nothing," IEEE Database Eng. Bull. 9.1 (1986), pp.4–9, 1986.

Masafumi Kinoshita received the M.E. degree in mechanical engineering from Nagoya University, Japan in 2002. He joined the System Development Laboratory, Hitachi, Ltd., Japan in 2002. He is currently a Senior Researcher at the Yokohama Research Laboratory, Hitachi, Ltd. His research interests are network architectures for enterprise and carrier systems, especially messaging systems. He is a member of IEICE.

Osamu Takada received his B.Sc. and M.E. degrees from Hokkaido University, Sapporo, Japan in 1977 and 1979, respectively. In 1979, he joined Hitachi, Ltd., as a member of the System Development Laboratory, Kawasaki. He has been engaged in the research and development of high speed LAN, packet switch and router, ITS applications and active networks with video transcoding. He received his Ph.D. degree in engineering from Tokyo University of Agriculture and Technology in 2013. Currently,

he is a member of the Yokohama Research Laboratory, Hitachi, Ltd. His research interests include network performance and availability, and network virtualization. He is a member of IEICE/CS.

Izumi Mizutani received the M.E. degree in information science from Nara Institute of Science and Technology in 2010. She joined the Central Research Laboratory, Hitachi, Ltd., Japan in 2010. She currently works as a researcher at the Yokohama Research Laboratory, Hitachi, Ltd. Her research interest is in carrier systems, especially messaging systems.

Takafumi Koike received the B.S degree in computer science and engineering from the University of Texas at Arlington, U.S. in 2005. He joined the Network Solution Department, Hitachi, Ltd., Japan in 2005. He is currently an Engineer of the TMS Solution Department, Hitachi, Ltd. His fields of work are on network and communication software, especially for messaging systems.

Kenji Leibnitz received his master and Ph.D. degrees in information science from the University of Würzburg in Germany. In May 2004, he joined Osaka University, Japan, as a Postdoctoral Research Fellow and from July 2006 he was a Specially Appointed Associate Professor at the Graduate School of Information Science and Technology. Since April 2010 he is a Senior Researcher at the National Institute of Information and Communications Technology (NICT), as well as a Guest Associate Professor at Osaka University and from April 2013 he is with the Center of Information and Neural Networks (CiNet) of NICT and Osaka University.

His research interests are in modeling and performance analysis of communication networks, especially the application of biologically and brain inspired mechanisms to self-organization in future networks.

Masayuki Murata received the M.E. and D.E. degrees in Information and Computer Science from Osaka University, Japan, in 1984 and 1988, respectively. In April 1984, he joined Tokyo Research Laboratory, IBM Japan, as a Researcher. From September 1987 to January 1989, he was an Assistant Professor with Computation Center, Osaka University. In February 1989, he moved to the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. In April

1999, he became a Professor of Cybermedia Center, Osaka University, and he is now with the Graduate School of Information Science and Technology, Osaka University, since April 2004. He has more than five hundred papers of international and domestic journals and conferences. His research interests include computer communication network architecture, performance modeling and evaluation. He is a member of IEEE, ACM and IEICE. He is a chair of IEEE COMSOC Japan Chapter since 2009. Also, he is now partly working at NICT (National Institute of Information and Communications Technology) as Deputy of New- Generation Network R&D Strategic Headquarters.