PAPER *Special Section on Parallel and Distributed Computing and Networking*

A Replication Protocol Supporting Multiple Consistency Models without Single Point of Failure

Atsushi OHTA[†a)], ***Nonmember***, **Ryota KAWASHIMA**[†b)], ***and*** **Hiroshi MATSUO**[†c)], ***Members***

SUMMARY Many distributed systems use a replication mechanism for reliability and availability. On the other hand, application developers have to consider minimum consistency requirement for each application. Therefore, a replication protocol that supports multiple consistency models is required. Multi-Consistency Data Replication (McRep) is a proxy-based replication protocol and can support multiple consistency models. However, McRep has a potential problem in that a replicator relaying all request and reply messages between clients and replicas can be a performance bottleneck and a Single-Point-of-Failure (SPoF). In this paper, we introduce the multi-consistency support mechanism of McRep to a combined state-machine and deferred-update replication protocol to eliminate the performance bottleneck and SPoF. The state-machine and deferred-update protocols are well-established approaches for fault-tolerant data management systems. But each method can ensure only a specific consistency model. Thus, we adaptively select a replication method from the two replication bases. In our protocol, the functionality of the McRep's replicator is realized by clients and replicas. Each replica has new roles in serialization of all transactions and managing all views of the database, and each client has a new role in managing status of its transactions. We have implemented and evaluated the proposed protocol and compared to McRep. The evaluation results show that the proposed protocol achieved comparable throughput of transactions to McRep. Especially the proposed protocol improved the throughput up to 16% at a read-heavy workload in One-Copy Serializability. Finally, we demonstrated the proposed failover mechanism. As a result, a failure of a leader replica did not affect continuity of the entire replication system unlike McRep.

key words: *distributed database, data replication, consistency model*

1. Introduction

In recent years, variant web services provided by Google, Amazon and Facebook have been popularized. Thereby, such large-scale services need to improve their availability and reliability for increasing demands of clients. The traditional single storage server cannot meet their demands, and therefore, a replication is widely used in distributed systems for reliability and availability. On the other hand, application developers have to consider minimum consistency requirement for each application. If the replication system provides only a short range of consistency models, an application suffers higher overheads of guaranteeing a stronger consistency than required. Therefore, a replication protocol that supports multiple consistency models is required.

Multi-Consistency Data Replication (McRep) [1] is a proxy-based replication protocol and can support six consistency models [2]–[8]. In McRep protocol, a *replicator* node relays all request and reply messages between clients and replicas. The replicator serializes transactions and resolves concurrency conflictions among transactions by managing all view differences of the database. Even though McRep provides a wide range of consistency guarantees, McRep has a problem in that the replicator becomes a performance bottleneck and a Single-Point-of-Failure (SPoF).

A state-machine replication [9] and a deferred-update replication [10] are well-established approaches for fault-tolerant data management systems. In both methods, replicas order requests by atomic broadcast [11] and process each request independently. The state-machine replication adopts a pessimistic approach, where each request is first ordered and all requests are executed in the same order (sequentially) on all replicas. The deferred-update replication relies on optimistic concurrency control [12]. Replicas atomically broadcast data items only when commit requests are issued. Therefore, performance can be improved in read-heavy workload. However, these two replication protocols only support single consistency model respectively.

We have proposed two types of replication protocols so far. The first work [13] improves performance of read-only transactions by placing the replicator to backend of replicas. But the SPoF still remains. The other work [14] introduces the multi-consistency support feature of McRep into the deferred-update protocol to avoid the SPoF. But this protocol cannot support Linearizability consistency model. In addition, the paper does not provide a concrete recovery mechanism.

In this paper, we propose an extended version of the work [14] to support the same consistency models of McRep and provide a concrete failure recovery protocol. In the privious paper, Linearizability was not implemented (and evaluated). In addition, the previous paper did not include the concrete recovery mechanism and its evaluation result. In the proposed protocol, any central proxy node is not used to prevent the SPoF, and the role of McRep's replicator is played by clients and replicas together. In addition, the state-machine and the deferred-update based replications are adaptively used to support the same consistency models as McRep. Each replica additionally serializes all transactions and manages all views of the database. Likewise, each client additionally manages status of its transactions. The proposed protocol can support multiple consistency models be-

Manuscript received January 6, 2016.
Manuscript revised May 22, 2016.
Manuscript publicized August 24, 2016.

[†]The authors are with Nagoya Institute of Technology, Nagoya-shi, 466–8555 Japan.
a) E-mail: a-ohta@matlab.nitech.ac.jp
b) E-mail: kawa1983@nitech.ac.jp
c) E-mail: matsuo@nitech.ac.jp

DOI: 10.1587/transinf.2016PAP0014

cause each replica knows which view of the database can ensure the required consistency model based on the deferred-update protocol. Furthermore, the state-machine protocol that provides a global view of the database is newly incorporated to support Linearizability. This paper also provides a concrete recovery mechanism and a result of a proof-of-concept evaluation of the mechanism.

The rest of this paper is organized as follows. In the next section, we explain the consistency models. An overview of McRep is given in Sect. 3. The proposed protocol is presented in Sect. 4. Section 5 shows the evaluation results. Finally, we summarize the conclusion and give future work in Sect. 6.

2. Consistency Models

The notion of consistency is a key factor to replication systems. In general, application developers have to consider minimum consistency requirement for each application to avoid unnecessary overheads. Followings are some specific consistency models of the most popular class of correctness criteria [1].

- Linearizability [2], [3] requires a total ordering on the transactions among the replicas even if a transaction does not access the conflicting data items. Linearizability preserves the real-time order for non-concurrent transactions, but a concurrent transactions can be executed in any total order between them.
- Sequential Consistency [4] requires a total ordering on the transactions among the replicas with a real-time ordering only on the non-concurrent transactions from the same clients. Hence, the same clients always observe a linearized view of the system.
- One-Copy Serializability [5] allows any total ordering in the transactions among the replicas. Real-time ordering is not needed in transactions. Hence, the clients can observe inconsistent responses from their consecutive requests due to stale data.

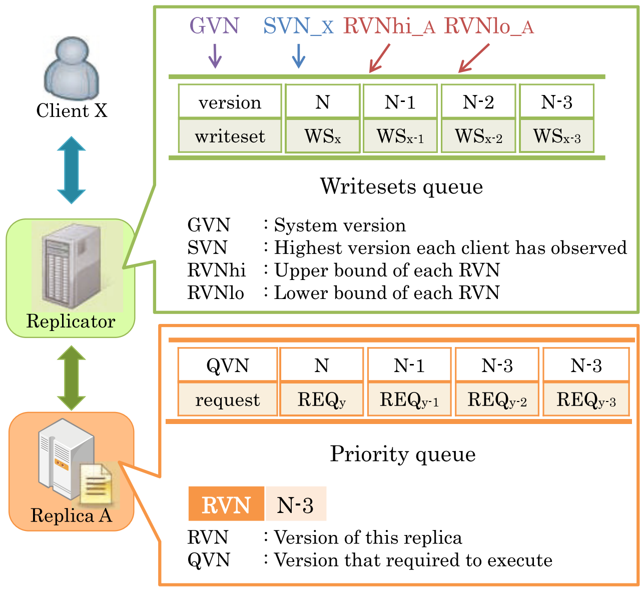
The stronger the consistency model, the more convenient to use. For example, Linearizability is even for formal verification because it preserves real-time ordering operations [15]. However, the stronger consistency models are not suitable for applications that require low latency because they need more costs of synchronization. Thereby, if a replication system supports multiple consistency models, the applications can choose an appropriate model without paying excess overhead.

3. The McRep Protocol

This section outlines characteristics of McRep, such as a proxy-based replication protocol and multi-consistency support feature, and discusses its problems.

3.1 Synopsis

McRep can support six consistency models (Linearizability, Sequential Consistency, One-Copy Serializability, Session Snapshot Isolation [7], Generalized Snapshot Isolation [8], and Causal Consistency [6]). Fundamental structure of McRep is based on Simple Replica Control Algorithm (SRCA) [16]. SRCA has a centralized server which forwards each transaction request to one of the replicas for execution. The centralized server also retrieves a writeset (a set of added, deleted and modified data items) of the transaction from reply messages to resolve every conflict among concurrent transactions. As shown in Fig. 1, McRep also has a centralized server (replicator) that relays both request and reply messages of transactions and resolves every conflict among concurrent transactions.


GVN
SVN_X
RVNhi_A
RVNlo_A
Client X
Replicator
Replica A
Writesets queue
GVN : System version
SVN : Highest version each client has observed
RVNhi : Upper bound of each RVN
RVNlo : Lower bound of each RVN
Priority queue
RVN N-3
RVN : Version of this replica
QVN : Version that required to execute

version	N	N-1	N-2	N-3
writeset	WS_x	WS_{x-1}	WS_{x-2}	WS_{x-3}

QVN	N	N-1	N-3	N-3
request	REQ_y	REQ_{y-1}	REQ_{y-2}	REQ_{y-3}

Fig. 1 Components of McRep

In practice, the replicator accumulates writesets with a sequential number that denotes a version of the system state in its queue. With these sequential numbers, the replicator manages following four types of version numbers that are represented indices into the queue to control consistency of replicas. Global Version Number (GVN) denotes the current version of the whole system state. Session Version Number (SVN) used for each client denotes a highest version of observed completed transactions. The replicator does not know the exact value of a Replica Version Number (RVN) which gives a latest system version at each replica due to asynchronous communications among replicas. Instead, the replicator manages both lower and upper bounds of RVN (RVNlo and RVNhi). The replicator computes the reQuired Version Number (QVN) as shown in Table 1 and tags the transaction request with the value. The version numbers are used for selecting an appropriate replica that can ensure a required consistency model. For example, if the required consistency model is sequential consistency, the QVN must be equal to SVN and the replicator forwards a transaction request to a replica whose RVNhi is greater than the QVN.

Each replica maintains a priority queue which contains

Table 1 Each QVN corresponding to a consistency model

Consistency model	QVN
Linearizability	GVN
Sequential consistency	SVN
One-Copy Serializability	0
Session Snapshot Isolation	SVN
Generalized Snapshot Isolation	0
Causal Consistency	SVN

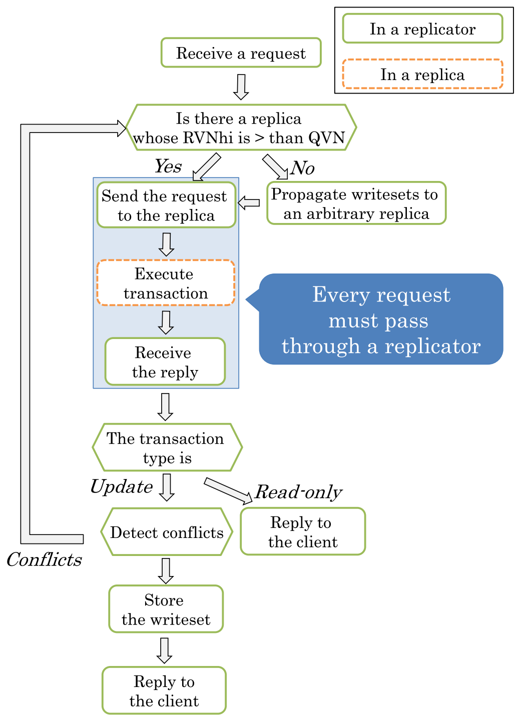


Fig. 2 A processing flow of the McRep

transaction requests sorted by their QVN order to exactly execute transactions at updated database. In precisely, a replica has to wait execution of a transaction until the RVN of the replica becomes greater than or equal to the QVN of the transaction request.

3.2 Processing Flow

A processing flow of McRep is shown in Fig. 2. The replicator must handle every transaction request because only the replicator maintains version numbers. When the replicator receives a transaction request, it computes the QVN of the request to select a replica whose RVNhi is greater than or equal to the QVN. If an appropriate replica is found, the replicator forwards the transaction request to the replica. Otherwise, the replicator needs to propagate the writesets to another replica, and the replicator forwards the transaction request to the replica.

A replica that has received writesets increments its own RVN such that RVN = GVN. Even though the replica is executing the forwarded transaction, the replica does not commit the transaction and the database itself is not changed at this point. In case of read-only transactions, the replicator simply relays reply messages from the replica to the client. Otherwise, the replicator accumulates the writeset of the transaction and replies to the client after resolving every conflict.

During the described processing, version numbers are updated as shown in Table 2.

Table 2 Updating version numbers in McRep

Replicator

Version	Timing	Updated value
RVNhi	Propagating writesets	GVN
RVNlo	Receiving a reply of a propagation	RVN
GVN	Storing a writeset to the queue	++GVN
SVN	Sending reply of a read-only transaction	max (SVN, RVN)
	Sending reply of an update transaction	GVN

Replica

Version	Timing	Updated value
RVN	Committing a writeset	RVN++

- RVNhi is updated when the replicator propagates writesets. The replicator updates RVNhi to GVN because the replicator propagates writesets that correspond to versions from RVNhi to GVN.
- RVN is updated when a replica commits the writesets. The replica increments RVN the same number of times as the number of received writesets.
- RVNlo is updated when the replicator receives a reply message of a propagation. The replicator updates RVNlo to RVN by the propagation.
- GVN is updated when the replicator stores a new writeset of a transaction because the new writeset indicates the latest global view of the database.
- Each SVN is updated when each client observes a new version. A client of a transaction observes the version of a replica that executes the transaction when the client receives a reply message. Therefore, the replicator updates SVN of the clients to RVN of the replica or current SVN. In the case of an update transaction, SVN is updated to GVN.

3.3 Problem

McRep is an asynchronous replication protocol and the transaction can be concurrently executed at each replica. In the McRep protocol, the replicator manages all important data for replication. Therefore, McRep has a potential problem in that the replicator becomes a performance bottleneck and a SPoF. Therefore, if the replicator downs, the entire system becomes unavailable until the replicator recovers from the halting state.

4. Proposed Replication Protocol without Single Point of Failure

This section presents details of the proposed replication protocol that can support multiple consistency models and avoid a SPoF.

4.1 Synopsis

We propose an extended replication protocol of work [14] by incorporating the state-machine protocol. In practice, the multi-consistency support mechanism of McRep is introduced into a combined state-machine and deferred-update replication protocols. Our protocol can support McRep-equivalent consistency models and additionally provide SPoF prevention feature. To avoid SPoF, the role of the McRep's replicator is moved to both replicas and clients. Each replica has to independently process transaction requests and ensure a required consistency model, and the functionality of replica is computation of QVN of transactions, serialization of transactions, retrieval of writesets, and management of the version numbers. Each client has a new role in management of their own version numbers (SVN).

The proposed protocol adaptively switches its replication method depending on a required consistency model. The state-machine based replication is used for Linearizability because replicas must know GVN before execution of transactions. The deferred-update based replication is used for the other five consistency models. The proposed protocol integrates atomic broadcast for the deferred-update and the state-machine based replication. Atomic broadcast is defined by the following properties in [17]:

- Validity: If a correct process p broadcasts m, then p eventually receives m.
- Agreement: For any two correct processes p and q, when any process s broadcasts m, if p receives m , then q eventually receive m.
- Integrity: For any message m, when any process s broadcasts m, every correct process p receive m at most once.
- Total order: If a correct process p receives m that a process s broadcasts after p receives m' that a process s' broadcasts, then every correct process q receives m only after it has received m'.

4.2 Version Control

In the proposed protocol, each replica and client control the aforementioned version numbers. Each replica manages not only its RVN, but also RVNhi of itself, RVNlo of all replicas, and GVN, and each client has to manage its own SVN.

- RVNhi is updated when a replica stores a new writeset of a transaction. This resembles to the update timing of McRep's GVN, but the replica increments its own RVNhi in the proposed protocol because the replica does not know exact status of the other replicas at this time. Thus, in the proposed protocol, each replica cannot obtain GVN like the McRep protocol. Instead, the proposed protocol can ensure Linearizability using the state-machine based replication.

Table 3 Updating version numbers in the proposal protocol

Replica

Version	Timing	Updated value
RVNhi	Storing a writeset to the queue	++RVNhi
RVN	Committing a writeset in the queue	++RVN
RVNlo	Receiving a broadcasted commit request	RVN
GVN	Determine the order of an execution	GVN

Client

Version	Timing	Updated value
SVN	Receiving a reply of a read-only transaction	max (SVN, RVN)
	Receiving a reply of an update transaction	RVNhi

- The update timing and the value of RVN is the same as McRep. But writesets are atomically broadcasted instead of the propagation process of the McRep's replicator.
- RVNlo's update process is totally different from McRep because a centralized server propagating writesets does not exist in the proposed protocol. RVNlo is updated when a replica receives an atomically broadcasted commit request from another replica. The commit requests contains an RVN value, and the replica updates source replica's RVNlo to the RVN value.
- GVN is not directly used in the proposed protocol. Instead, we use the atomic broadcast protocol [11] to order execution of transactions, which is equivalent to the GVN based ordering.
- SVN's update timing and updated value is the same as McRep. However, each SVN is updated by each client and the client observes RVNhi instead of GVN at an update transaction. The replica replies updated RVN at a read-only transaction or updated RVNhi at an update transaction.

The above update processes of version numbers are summarized in Table 3.

4.3 Processing Flow

In the proposed protocol, the lifetime of a transaction is divided in two phases: the execution phase and the termination phase.

4.3.1 Execution Phase

Figure 3 describes an execution flow of a transaction. When a replica receives a transaction request from a client, the replica computes the QVN of the transaction based on the required consistency model (1). Calculation method of QVN varies depending on the required consistency model.

Linearizability

The replica atomically broadcasts the transaction to obtain GVN.

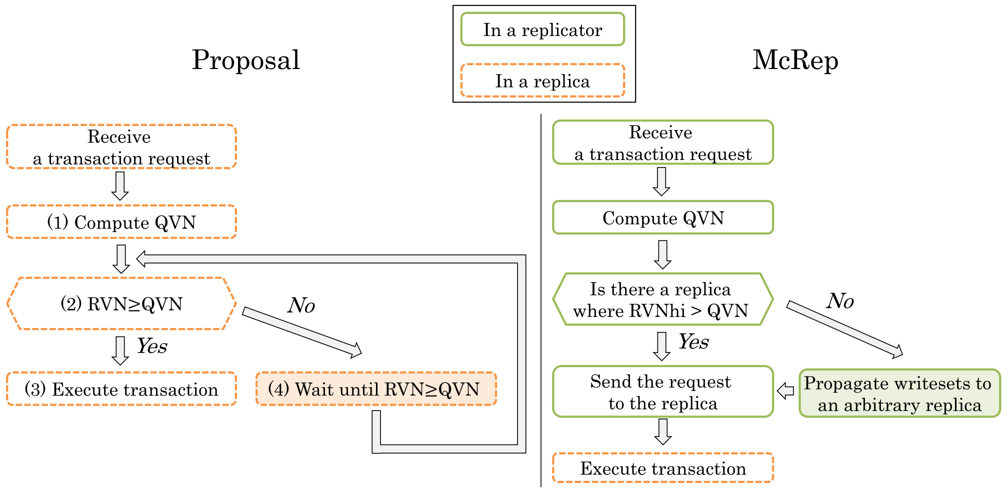

In a replicator
In a replica
Proposal
Receive a transaction request
(1) Compute QVN
(2) RVN≥QVN
No
Yes
(3) Execute transaction
(4) Wait until RVN≥QVN
McRep
Receive a transaction request
Compute QVN
Is there a replica where RVNhi > QVN
No
Yes
Send the request to the replica
Propagate writesets to an arbitrary replica
Execute transaction

Fig. 3 Execution flows of a transaction's lifetime in the proposed and McRep protocols

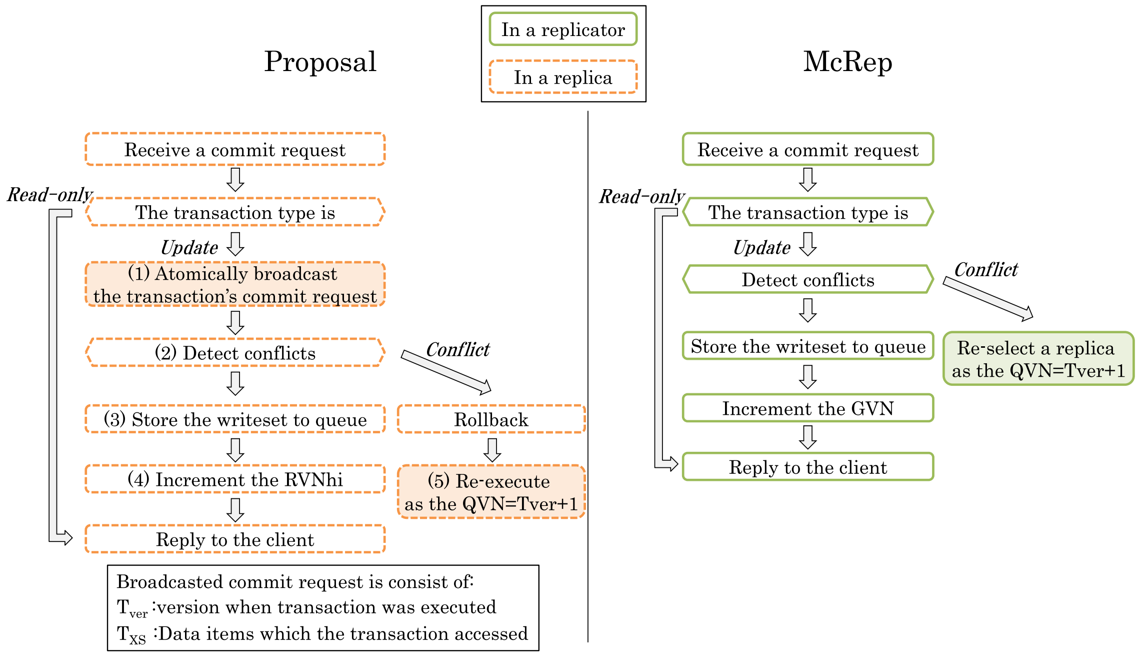

In a replicator
In a replica
Proposal
Receive a commit request
Read-only
The transaction type is
Update
(1) Atomically broadcast the transaction's commit request
(2) Detect conflicts
Conflict
(3) Store the writeset to queue
Rollback
(4) Increment the RVNhi
(5) Re-execute as the QVN=Tver+1
Reply to the client
Broadcasted commit request is consist of:
T_ver :version when transaction was executed
T_XS :Data items which the transaction accessed
McRep
Receive a commit request
Read-only
The transaction type is
Update
Detect conflicts
Conflict
Store the writeset to queue
Re-select a replica as the QVN=Tver+1
Increment the GVN
Reply to the client

Fig. 4 Termination flows of a transaction's lifetime in the proposed and McRep protocols

Sequential Consistency, Session Snapshot Isolation and Causal Consistency

The replica retrieves SVN from the transaction request.

One-Copy Serializability and Generalized Snapshot Isolation

The replica does nothing because QVN is zero.

Next, the replica compares its own RVN to QVN of the transaction (2). If RVN is greater than or equal to QVN, the replica straightforwardly executes the transaction (3). Otherwise, the execution is postponed until the RVN is updated by an atomic broadcast from another replica (4).

4.3.2 Termination Phase

The termination phase starts when the client requests the replica to commit the transaction. This phase is skipped when the required consistency model is Linearizability because any concurrency conflicts do not occur for the serialization before the execution.

A process of the termination phase differs depending on the transaction type as shown in Fig. 4. When a client requests a replica to commit a read-only transaction, the replica commits the transaction and sends a reply message to the client. Otherwise, the replica atomically broadcasts the commit request to all replicas to uniquely determine the order of the commitment (1). The commit request consists of data items accessed by the transaction (T_{XS}) and the RVN value of the transaction (T_{ver}). When the commit request is delivered to the replica, the replica detects concurrency conflicts between T_{XS} and writesets of concurrent transactions in the queue in the same manner as McRep (2). If no conflict is detected, a writeset of T_{XS} is stored in the writeset queue (3) and RVNhi is updated to point to the newly stored element of the queue (4). Next, the replica immediately replies to the client. At this time, the transaction successfully terminates. If any conflict occurs, the transaction is rollbacked because this replica has already executed the transaction. Next, the replica retries to execute the transaction with newer version numbers to avoid conflicts (i.e. T_{ver} + 1) (5). When the number of the retries exceeds a threshold (X times), the transaction is aborted and the replica notifies the conflict to the client.

4.4 Failure Recovery Mechanism

When a replica recovers from a crash, the replica cannot participate in the request processing until the replica retrieves writesets of transactions that successfully completed before the recovery. If the replica immediately participate in the processing before the retrieval, transactions can observe inconsistent view of the database. Therefore, the proposed recovery protocol is based on a standard recovery protocol from Viewstamped Replication [18]: it requires $2f + 1$ replicas and provides safety as long as no more than f replicas fails simultaneously. The recovery protocol is explained as follows:

1. The recovering replica, i, sends a <Recovery i, l, h> message to the other replicas, where l and h are the lowest and highest bounds of the version number of the writesets to retrieve.
2. A replica j replies to the Recovery message. The replica sends a <RecoveryResponse j, l, h, w> message to the recovering replica, where w is its writesets whose version numbers are from l to h.
3. The recovering replica waits to receive at least 1 RecoveryResponse messages from different replicas. The recovering replica updates its writeset queue if l and h are matches between Recovery and RecoveryResponse.

4.5 Correctness

The requirements of the proposed protocol is following.

- support multiple consistency models.
- if the failure occurs at one replica, the system works correctly.

4.5.1 Correctness of Ensuring Consistency Models

First, we argue why the proposed protocol can support multiple consistency models. A common requirement of implemented consistency models (One-Copy Serializability, Linearizability and Sequential Consistency) is that every replica observes the same total ordering on the transaction [1]. Different consistency model requires different total ordering on the transaction:

(1) One-Copy Serializability

A required condition for One-Copy Serializability is that transactions are serializable. Then, we argue whether the proposed protocol can ensure that all committed transactions are serializable. The proposed protocol checks conflicts between concurrent update transactions and rollbacks the execution of transaction if conflicts occur. In the proposed protocol, all transactions are optimistically executed at any replica. So, update transactions may conflict with concurrent transactions. If conflicts occur, transactions cannot serializable. So, the system must uniquely determine followings among all replicas: (a) which transactions are concurrent and occur conflicts and (b) which transaction is rollbacked. Suppose that a transaction T_i conflicts with a transaction T_j. T_i is rollbacked if:

- $start(T_i) < end(T_j) < end(T_i)$
- $WS(T_j) \cap XS(T_i) \neq \emptyset$

Where $start(T)$ is the time starting the execution of a transaction T, $end(T)$ is the time completing the termination of a transaction T, $WS(T)$ is a writeset of a transaction T, and $XS(T)$ is a readset and a writeset of a transaction T. The first inequality tests the concurrency of T_i and T_j and the second inequality tests conflicts between T_i and T_j. The system must determines an total order of $end(T_i)$ and $end(T_j)$ among replicas to implement the first test. So, replicas atomically broadcasts termination requests to all replicas because the atomic broadcast definition ensures the total ordering on all broadcasted messages [17]. To implement the second test, every replica has a queue that contains writesets of formerly committed transactions and checks conflicts on each termination request. Consequently, in the proposed protocol, the system can determine (a) which transactions are concurrent and occur conflicts and (b) which transaction is rollbacked by atomic broadcast and the writeset queue.

(2) Sequential Consistency

Required conditions for Sequential Consistency are (a) transactions are serializable and (b) transactions are executed with the highest version observed by the same client. The condition (a) is satisfied as well as One-Copy Serializability. We argue whether the proposed protocol satisfy the condition (b). Suppose that a client send a transaction T_j after a last update transaction T_i and T_i create a new version V_i. Sequential Consistency can ensure if:

- $end(T_i) < create(V_i) < start(T_j)$

Where $create(V_i)$ is the time that the database is changed by a modification of T_i. To ensure Sequential Consistency, $start(T_j)$ must follow $create(V)$. So, a replica wait an execution of T_j until a version of this replica is equal to V_i. For the replica to know that V_i is the highest version observed by the client, the client tells the replica V_i at the time that the client sends T_j a replica tells the client that V_i is the version created by T_i at $end(T_i)$. In the proposed protocol, when a replica sends a reply of a transaction, the replica sends the version observed by the transaction. Hence, clients know the highest version observed by itself, and clients tell the version to any replica. Consequently, in the proposed protocol, any replica ensures that transactions are executed with the highest version observed by a client.

(3) Linearizability

Required conditions for Linearizability are (*a*) transactions are serializable and (*b*) transactions are executed in a replica with zero version divergence, i.e. with the latest system version. The condition (*a*) is satisfied as well as One-Copy Serializability. We argue whether the proposed protocol satisfy the condition (*b*). Suppose that a request of a transaction T_{i+1} received and the latest system version is V_i. Linearizability can ensure if:

- $create(V_i) < start(T_{i+1})$

So, a replica must execute a transaction creating V_i before $start(T_{i+1})$. In the proposed protocol, replicas atomically broadcast all transactions before their execution, then replicas execute all transactions and the transactions create a new system version in the broadcasted order. Hence, a transaction that will create the version V_i execute before $start(T_{i+1})$ Consequently, in the proposed protocol, all transactions can execute in replicas with the latest system version.

4.5.2 Correctness of Recovery Protocol

We discuss the correctness of the recovery protocol. We assume that the only way replicas fail is by crashing (crash-stop model), so that a machine is either functioning correctly or completely stopped. The proposed protocol does not handle Byzantine failures. In the proposed protocol, $f + 1$ replicas are always correct and have the latest view of the database by atomic broadcast. Thus, correctness of the proposed protocol depends on the correctness of the recovery protocol. In precisely, the recovering replica must not participate in the request processing at an inconsistent view of the database. The recovery protocol is correct because it guarantees that a recovering replica participates in the request processing after the replica retrieves writesets that the replica lost in the down time. When a replica recovers, it doesn't know latest writesets. However, when the replica receives a response for its Recovery message, the replica retrieves the writesets from the lowest bound l (the last committed writeset when the replica failed) to the highest bound h (the first broadcasted writeset when the replica recovers). Consequently, the recovery protocol ensures the replica learns the latest state of queue.

5. Evaluation

We implemented the proposed replication protocol with PostgreSQL [19] as replicas. We also implemented McRep with pgpool-II [20] as the replicator because the existing study [1] only conducted a simulation-based evaluation for McRep. In the proposed protocol, we implemented the atomic broadcast protocol (Paxos [21]) with LibPaxos [22] on each replica.

Table 4 The specification of PostgreSQL, pgpool-II and client nodes

	PostgreSQL	pgpool-II
version	9.3.5	3.3.3
OS	Linux 3.5.0-23-amd64 Ubuntu 12.04 Server	
CPU	Intel(R) Core(TM) i5 3470 @ 3.2GHz	
Memory	16GB	
Network	1000BASE-T	

5.1 Evaluation Environment

Machine specifications of PostgreSQL, pgpool-II and client nodes are shown in Table 4. We used a pgbench benchmark program [23] with three workloads consisting of update transactions and read-only transactions at a certain ratio. This benchmark program creates a table that has 100,000 entries. The update transactions randomly update an entry of the table. The read-only transactions randomly select and read an entry of the table. We examined three types of workloads that have different ratio of read-only and update queries as follows.

Workload A:	read-only (90%) / update (10%)
Workload B:	read-only (50%) / update (50%)
Workload C:	read-only (10%) / update (90%)

In the following evaluations, we measured the throughputs of transactions for both the protocols during one minute, and in the proposed protocol, a leader replica was selected by a Paxos consensus protocol and only the leader had a role in atomically broadcasting to optimize the consensus protocol.

5.2 Performance Effect of the Number of Clients

We evaluated the performance of both protocols with changing the number of clients (50–400), and the number of replicas was fixed to six. We evaluated the throughput at Linearizability, Sequential Consistency and One-Copy Serializability consistency models[†]. Figure 5 shows the throughputs of transactions. In Sequential Consistency and Linearizability, executions of transactions have to wait both writeset arrival and writeset commitment to the database. As a result, the throughputs were not scaled as the increase of the number of clients in both protocols. In One-Copy Serializability, their throughputs were increased as the growing the number of clients because the waiting period was omitted to execute transactions in both protocols. In McRep, however, the throughputs were peaked at 200 clients because the replicator became a performance bottleneck, while in the proposed protocol, its throughput still increased at 400 clients. In both protocols, fundamental throughputs decreased as increasing the ratio of update transactions. The throughput decline is

[†]Performance of the other consistency models was the same with that of the above three consistency models. Thus, this paper does not provide the performance of the other consistency models.

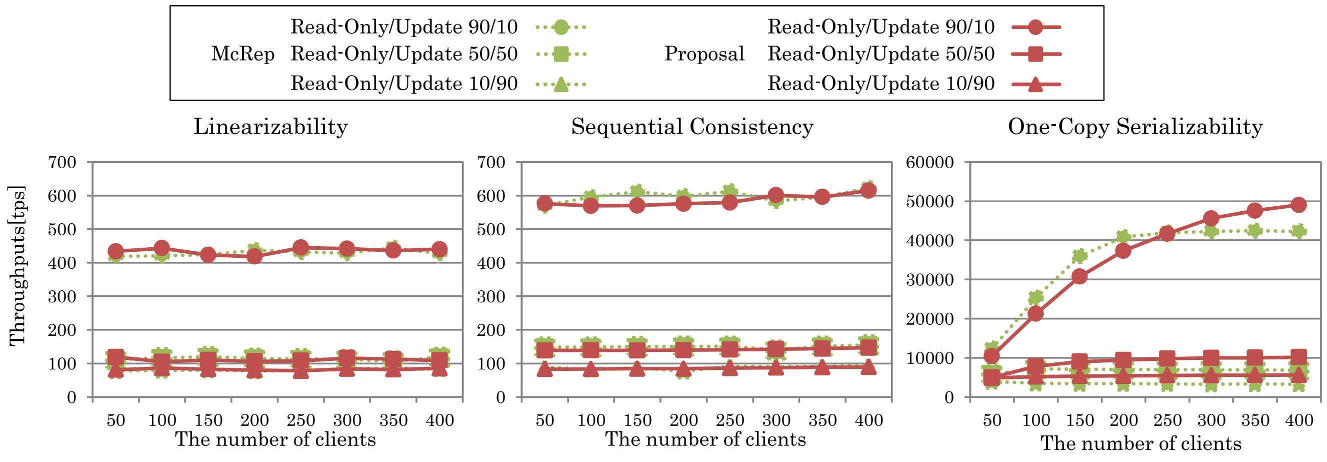


Fig. 5 A comparison of throughputs of different number of clients in two consistency models between McRep and the proposed protocol, where 10/90, 50/50 and 90/10 denote the ratio of read-only and update transactions.

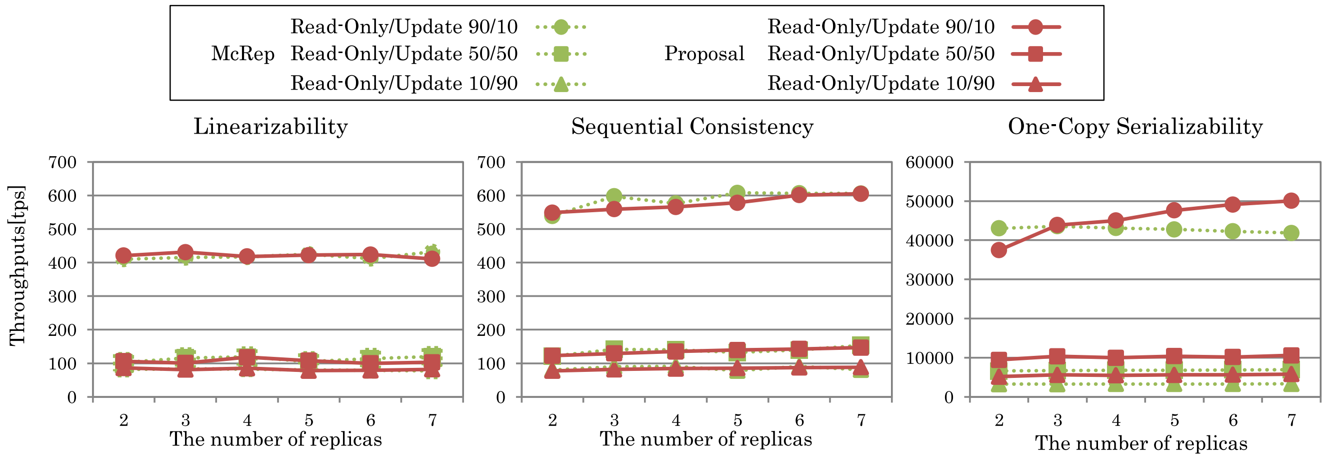


Fig. 6 A comparison of throughputs of different number of replicas in two consistency models between McRep and the proposed protocol, where 10/90, 50/50 and 90/10 denote the ratio of read-only and update transactions.

a result of the conflict detection. In McRep, the replicator has to handle all the update transactions, and this results in heavy access cost of the shared memory of the propagation queue. In the proposed protocol, the decline of performance is mainly a result of the cost of atomic broadcasts because the processing of transaction requests was distributed to each replica. Therefore, the proposed protocol can prevent the performance loss if the cost of atomic broadcast is reduced by using an optimized atomic broadcast protocol depending on the reliability of the network and the server nodes.

In Sequential Consistency and Linearizability, executions of transactions have to wait both writeset arrival and writeset commitment to the database. Figure 7 shows detailed average latencies of update transactions at each consistency model at read-heavy workload. The *consensus* in Fig. 7 means the time for completing atomic broadcast of a transaction. This is almost as same as the three consistency models. The *waiting period* before an execution of a transaction takes much more time than the consensus because the

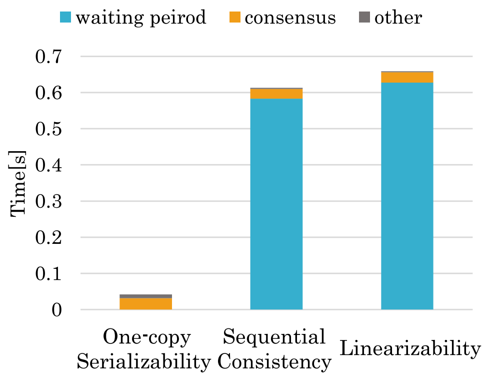


Fig. 7 Details of average latencies of update transactions.

writing modifications of writesets to a disk takes time.

5.3 Performance Effect of the Number of Replicas

We evaluated the performance of both protocols with chang-

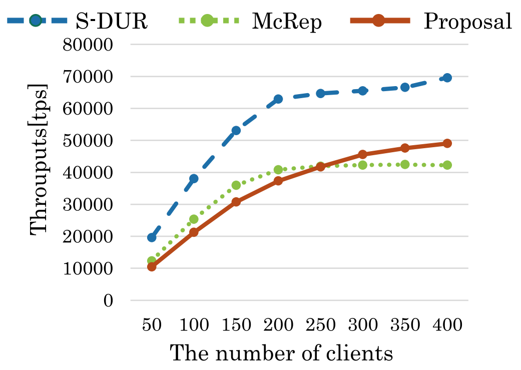


Fig. 8 The throughput of S-DUR with 100 % of local transactions comparing with the proposed protocol and McRep in One-Copy Serializability.

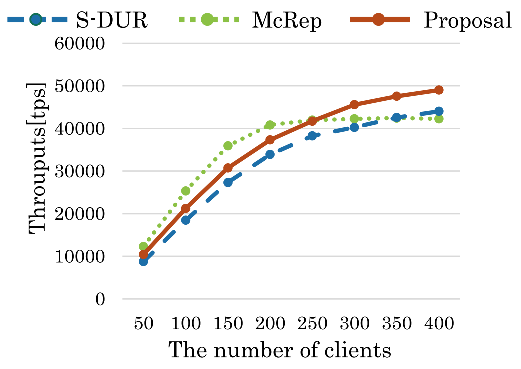


Fig. 9 The throughput of S-DUR with 100 % of global transactions comparing with the proposed protocol and McRep in One-Copy Serializability.

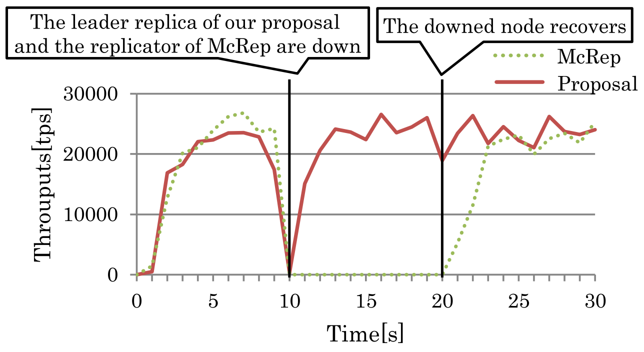


Fig. 10 Changes in transaction request throughputs during a replica or a replicator failover.

ing the number of replicas (2–7) and the number of clients was fixed to 400. Figure 6 shows the throughputs of both protocols. In Sequential Consistency and Linearizability, the throughputs remained flat regardless of the number of replicas. The reason is the same as described in Sect. 5.2. In One-Copy Serializability, any throughput gain was not seen in McRep because the replicator became a performance bottleneck. On the other hand, in the proposed protocol, throughputs of the workload A rose to only 1.3 times as increasing the number of replicas from two to seven. This result is ascribed to the increasing of the cost of the atomic broadcast as the number replica increases. However, the throughputs of the proposed protocol were better than or comparable to those of McRep in any cases.

5.4 Comparing the Performance with an Extension to the Deferred-Update Replication

We compared the performance with an extension to the deferred-update replication *scalable deferred update replication (S-DUR)* [24]. The key insight of S-DUR is to divide the database into partitions, and replicate each partition among a group of replicas. This protocol eliminates communications among replicas that have different partitions for transactions that access single partition (local transaction). But, extra communication is needed to terminate transactions that access multiple partitions (global transaction). Therefore, the throughput of local transactions scales out as the number of partitions increases but the throughput of global transactions is worse than that of the classical deferred-update.

We evaluated the throughput of local transactions and global transactions in S-DUR under the read-heavy workloads with changing the number of clients (50–400). The number of replicas was fixed to six. We divided the database into three partitions for S-DUR (two replicas per partition). First, each transaction accessed single partition for S-DUR. This environment is ideal for S-DUR. Figure 8 shows the throughputs of S-DUR, the proposed protocol and McRep. The throughput of S-DUR was the best of the three protocols because S-DUR can eliminate communications across different partitions for local transactions. Second, each transaction accessed all partitions. Figure 9 shows the throughputs of S-DUR, the proposed protocol and McRep. This is the worst case for S-DUR because communications between partitions is needed on each transaction than the classical deferred-update replication. To check whether transactions is serializable, global transactions need the additional two-phase commit-like communications between partitions after transactions atomically broadcasts to servers in each partition. The throughput of S-DUR was slightly worse than the proposed protocol because the proposed protocol uses the classical deferred-update replication. Consequently, if the transactions access single partition, S-DUR improves the throughput as the number of partitions. However, the proposed protocol can substitute S-DUR for the classical deferred-update replication to support One-Copy Serializability. And the prposed protocol can ensure the multiple consistency models but S-DUR supports only One-Copy Serializability.

5.5 Evaluation of Failover

Finally, we conducted a proof-of-concept evaluation of the recovery mechanism of the proposed protocol. In this experiment, the number of replicas and clients were fixed to seven and 100 respectively, and the required consistency model was One-Copy Serializability. Note that the recovery mechanism works same way for the other consistency models.

During the experiment, the leader replica of our proposal and the replicator of McRep were down at tenth second and both of them recovered at 20th second.

Figure 10 shows the throughputs of transaction request during the experiment. In McRep, the throughputs became zero during the replicator is down. On the other hand, in the proposed protocol, the throughputs instantaneously dropped when the leader node downs but recovers in 0.87 seconds. This was the time taken for choosing a new leader with a consensus protocol. When the downed node recovers, the significant losses of throughput cannot be seen at the time because the recovery protocol and usual request processing were concurrently performed.

6. Conclusion

In this paper, we have proposed a novel replication protocol supporting McRep-equivalent consistency models without a SPoF that McRep suffers. McRep places a proxy-based centralized replicator, and therefore, McRep potentially has a performance bottleneck and the SPoF. On the other hand, our proposed replication protocol resolves these problems by eliminating the centralized proxy without losing the entire system functionality of McRep. The evaluation results show that the proposed protocol achieved comparable throughput of transactions to McRep. Especially, the proposed protocol gained the throughput of transactions with 400 clients at a read-heavy workload in One-Copy Serializability, while that of McRep peaked with 200 clients. We have also proposed a concrete recovery mechanism and proved that the fast failover mechanism works effectively.

As future work, we are planning to design a partial-replication protocol for each consistency model to improve performance of the entire system. Besides, we consider introduction of a yet another communication protocol that do not require a guarantee of total-order of transactions.

Acknowledgments

This work was supported in part by MEXT KAKENHI Grant Number 24500113 and 15K00168.

References

[1] R. Al-Ekram and R. Holt, "Multi-consistency Data Replication," Proc. IEEE 16th Int'l Conf. on Parallel and Distributed Systems (ICPADS), pp.568–577, Dec. 2010.

[2] M.P. Herlihy and J.M. Wing, "Linearizability: A Correctness Condition for Concurrent Objects," ACM Trans. Program. Lang. Syst., vol.12, no.3, pp.463–492, 1990.

[3] T. Riegel, C. Fetzer, H. Sturzrehm, and P. Felber, "From Causal to Z-linearizable Transactional Memory," Proc. the 26th Annual ACM Symp. on Principles of Distributed Computing, pp.340–341, Aug. 2007.

[4] L. Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," IEEE Trans. Comput., vol.28, no.9, pp.690–691, 1979.

[5] P.A. Bernstein and N. Goodman, "A proof technique for concurrency control and recovery algorithms for replicated databases," Distributed Computing, vol.2, no.1, pp.32–44, 1987.

[6] M. Raynal, G. Thia-Kime, and M. Ahamad, "From serializable to causal transactions for collaborative applications," Proc. EUROMICRO 23rd Conf. EUROMICRO 97. New Frontiers of Information Technology, pp.314–321, Sept. 1997.

[7] K. Daudjee and K. Salem, "Lazy Database Replication with Snapshot Isolation," Proc. 32nd Int'l Conf. on Very Large Data Bases, pp.715–726, 2006.

[8] S. Elnikety, W. Zwaenepoel, and F. Pedone, "Database Replication Using Generalized Snapshot Isolation," Proc. IEEE 24th Symp. on Reliable Distributed Systems, pp.73–84, 2005.

[9] F.B. Schneider, "Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial," ACM Comput. Surv., vol.22, no.4, pp.299–319, 1990.

[10] B. Charron-Bost, F. Pedone, and A. Schiper, Eds., "Replication: Theory and Practice," ser. LNCS, vol.5959, Springer, 2010.

[11] V. Hadzilacos and S. Toueg, "Distributed Systems (2Nd Ed.)," ch. Fault-tolerant Broadcasts and Related Problems, pp.97–145, 1993.

[12] H.T. Kung and J.T. Robinson, "On Optimistic Methods for Concurrency Control," ACM Trans. Database Syst., vol.6, no.2, pp.213–226, 1981.

[13] M. Masaya, A. Ohta, R. Kawashima, and H. Matsuo, "Back-end Based Data Replication Supporting Multiple Consistency Models," IPSJ SIG Technical Report, vol.2015-OS-132, no.14, pp.1–11, 2015 (in Japanese).

[14] A. Ohta, R. Kawashima, and H. Matsuo, "Deferred-Update Replication Supporting Multiple Consistency Models," 3rd Int'l Symp. on Computing and Networking (CANDAR), pp.610–612, 2015.

[15] H. Attiya and J.L. Welch, "Sequential consistency versus linearizability," ACM Transactions on Computer Systems (TOCS), vol.12, no.2, pp.91–122, 1994.

[16] Y. Lin, B. Kemme, M. Patiño Martínez, and R. Jiménez-Peris, "Middleware Based Data Replication Providing Snapshot Isolation," Proc. the ACM SIGMOD Int'l Conf. on Management of Data, pp.419–430, 2005.

[17] V. Hadzilacos and S. Toueg, "A modular approach to fault tolerant broadcasts and related problems," Tech. Rep., 1994.

[18] B. Liskov and J. Cowling, "Viewstamped replication revisited," Technical Report MIT-CSAIL-TR-2012-021, MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA, July 2012.

[19] "Postgresql." https://www.postgresql.jp/

[20] "pgpool-II." http://www.pgpool.net/

[21] L. Lamport, "Paxos made simple," ACM Sigact News, vol.32, no.4, pp.18–25, 2001.

[22] D. Sciascia, "LibPaxos." http://libpaxos.sourceforge.net/

[23] "pgbench." http://postgresql.org/docs/9.2/static/pgbench.html

[24] D. Sciascia, F. Pedone, and F. Junqueira, "Scalable deferred update replication," Proc. 42nd Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN), pp.1–12, 2012.

Atsushi Ohta was born in 1991. He graduated from Nagoya Institute of Technology in 2015. He has became a master student in Nagoya Institute of Technology in 2015.

Ryota Kawashima was born in 1983. He received his M.S. degree from Iwate Prefectural University in 2007 and also received Ph.D. degree from The Graduate University for Advanced Studies (SOKENDAI) in 2010. He had worked as a software engineer at ACCESS CO., LTD. and Stratosphere, Inc. He has became an assistant professor at Nagoya Institute of Technology in 2013. His research interest is Software-Defined Networking. He is a member of IEICE, IPSJ, and IEEE.

Hiroshi Matsuo was born in 1960. He received his M.S. in 1985 and also received Ph.D. degree in 1989 from Nagoya Institute of Technology. He became an assistant professor in 1989, lecturer in 1993, associate professor in 1995, and professor in 2003 at Nagoya Institute of Technology. His research interest is distributed cooperative system. He is a member of IEICE, IPSJ, and IEEE.