Fault Tolerance in the Traffic Management System of a Last-mile Transportation Service

Koji Hasebe, Shohei Sasaki and Kazuhiko Kato
Department of Computer Science, University of Tsukuba, 1-1-1, Tennodai, Tsukuba 305-8573, Japan

Keywords: Last-mile Transportation, Traffic Management System, Fault Tolerance, Passive Replication.

Abstract: A last-mile transportation system has previously been developed using semi-autonomous driving technologies. For traffic management of this system, travel requests are gathered at a central server and, on this basis, the operation schedule is periodically updated and distributed to each vehicle via intermediate servers. However, the entire traffic system may stop if the central server malfunctions owing to an unforeseen event. To address this problem, we propose a fault-tolerant mechanism for the traffic management system of a last-mile transportation service. We use a modified primary-backup (or so-called passive) replication technique. More precisely, one of the intermediate servers is chosen as the central server and the other intermediate servers periodically receive state-update messages from the central server, allowing them to update their state to match that of the central server. If the central server fails, one of the node servers is selected to take over as the new central server. The present paper also demonstrates the availability through failure patterns in experiments with a prototype implementation.

1 INTRODUCTION

Improvements to public transport are important in realizing the next-generation smart city. The convenience of basic public transportation infrastructure, such as railroads and large-scale facilities, is being improved but the last-mile transportation network is still underdeveloped. There is thus a need for the creation and social implementation of a new transportation system having high accessibility.

Last-mile transportation systems offer a convenient means of transportation, especially for elderly people and people with disabilities because a stop can be finely moved to where there is passenger demand. However, the transport network of such a system tends to be complicated, and there remains the problem that the speed of passenger transport is reduced by taking all passengers to all stops and by the transfer operation during the movement process.

To improve the situation described above, a last-mile public transportation system based on technologies of semi-autonomous driving has been developed (Hasebe et al., 2017b). The previously proposed system involved vehicles of two types, those that lead a fleet and are driven manually and those that are driverless and follow another vehicle. One of the major features of the system is the ability for vehicles to run in a fleet without physical connecting points, allowing a fleet of vehicles to be smoothly reorganized. With this feature, even if passengers with different destinations board the same fleet of vehicles, by appropriately detaching or reorganizing vehicles at branch points of the route (called *nodes*), the passengers can reach their destinations without transferring between fleets.

This transportation system aggregates the travel requests in real time and dynamically determines the schedule of the vehicles. To realize such scheduling, the system has a single central server and intermediate servers (called *node servers*) placed at each node to manage the traffic of the fleet by centralizing demand information, and the system instructs each vehicle from this central hub. However, the entire traffic system may stop if the central server malfunctions owing to some unforeseen event.

To address this problem, we propose in this paper a fault tolerant mechanism for the traffic management systems of last-mile transportation services. We use a primary-backup (or so-called passive) replication technique to make the central server redundant. More precisely, one of the node servers is chosen as the central server and processes requests from passengers and then decides a new schedule to be delivered to other node servers at regular intervals. The

Hasebe, K., Sasaki, S. and Kato, K.
Fault Tolerance in the Traffic Management System of a Last-mile Transportation Service.
DOI: 10.5220/0006794505520557
In *Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2018)*, pages 552-557
ISBN: 978-989-758-293-6

other node servers receive the schedule from the central server and give instructions to nearby vehicles. These servers also periodically receive state-update messages from the central server, allowing them to update their state to match that of the central server. If the central server fails, one of the node servers is selected to take over as the new central server.

We here use the replication technique introduced by (Hasebe et al., 2014) as the algorithm that realizes this primary-backup replication; i.e., instead of using the usual primary-backup technique based on the majority-based consensus algorithm, we use a modified Paxos algorithm that allows agreement to be reached by fewer than half the nodes. Using this replication technique, it is possible to continue traffic management even if the network is divided and the majority of the node servers fail.

In this paper, we also demonstrate availability through failure patterns in experiments with a prototype implementation.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 overviews the transportation system targeted in this study. Section 4 presents our passive replication technique for traffic management systems. Section 5 demonstrates the availability of the traffic management system given various patterns of failure through simulations. Finally, Section 6 concludes the paper and presents future work.

2 RELATED WORK

Most replication techniques that have been proposed so far can be classified into two main categories: state machine and primary-backup replications.

In state machine (i.e., active) replication (Schneider, 1990), each server processes requests from clients and transitions independently. Generally, each transition is coordinated across servers by means of a consensus algorithm. Although this technique is useful, it has two important drawbacks as a result of its low response time: a high computational cost and the need for client requests to be processed in a deterministic manner. Meanwhile, primary-backup replication is useful because of its low computational cost and applicability to nondeterministic services. However, as mentioned in Section 1, a mechanism that allows agreement on the current primary and its implementation is needed; this is not necessary in state machine replication systems. As explained above, the merits and demerits of these techniques are complementary. To counter these drawbacks, variant schemes have been proposed in recent years. These include semiactive replication (Stodden, 2007) and semipassive replication (Defago and Schiper, 2004).

Figure 1: Conceptual illustration of a fleet of passenger-carrying vehicles.

However, it is assumed that these techniques will employ a majority-based consensus. Thus, although the techniques guarantee replica consistency, they cannot tolerate the simultaneous failure of a majority of the nodes or partitioning of the network. To address the issue of availability, Dolev et al. (Dolev et al., 2010) proposed optimistic state machine replication based on a self-stabilizing consensus algorithm (Dijkstra, 1974; Dolev, 2000). Subsequently, they suggested a self-stabilizing primary backup replication technique and its application to Internet service platforms (Hasebe et al., 2011).

As shown above, various replication techniques have been proposed and widely used in distributed systems. However, application of the techniques to transportation systems has not been thoroughly investigated. One motivation of the present study is to show the usefulness of such a replication technique in the development of a transportation system.

3 OVERVIEW OF LAST-MILE TRANSPORTATION SYSTEMS

This section outlines the last-mile transportation system targeted in this study. (Also see the literature (Hasebe et al., 2017b; Hasebe et al., 2017a) for a more detailed explanation.)

3.1 Vehicles

The transportation system targeted in this study consists of vehicles that are able to travel in a row without having physical contact points. A conceptual illustration is shown in Fig. 1.

The vehicles are classified according to their function into two types.

- *Lead vehicles*: vehicles driven manually at the head of a fleet of vehicles.
- *Trailing vehicles* (or *trailers* for short): driverless vehicles that run autonomously behind another vehicle.

Vehicles can be arbitrarily connected through electronic control to a fleet as long as the fleet capacity is not exceeded.

3.2 Traffic Network

In the transportation system, as for existing railroad and bus networks, the travel routes connect a plurality of prescribed stops where passengers may board or alight. Additionally, in this travel route network, zones are provided depending on the geographic location where stops overlap with one or more zones. Usually, there are multiple stops in each zone and a travel route passing through all the stops in a zone is set. Here, a stop located in multiple zones (i.e., a stop at branch points in the traffic network) is called a *node*.

3.3 Reorganization of Vehicles

Lead vehicles can disconnect any trailers at a node. In addition, a location for a *vehicle pool* is provided at each node for the temporary parking of detached unmanned lead vehicles. Furthermore, a lead vehicle can be provided with any number of trailers from the vehicle pool, as long as the fleet capacity is not exceeded.

3.4 Vehicle Traffic Management

To realize our vehicle traffic management scheme, the operation schedule needs to be arranged according to the current requests and delivered to each vehicle. We here assume a certain centralized control. Specifically, we set up a server for vehicle operation management (hereinafter referred to as the *central server*) in the system's hub. The central server receives requests for travel from passengers using smartphones or other means of communication, minimizes passenger waiting time, determines a targeted optimum operation schedule, and instructs each lead vehicle.

Moreover, in addition to the central server, a server at each node (hereinafter referred to as a *node server*) is maintained. The configuration and the operation route of a fleet as determined by the central server are transmitted once to each vehicle via the node server within the communication network.

4 REPLICATION TECHNIQUE FOR A TRAFFIC MANAGEMENT SYSTEM

4.1 Overview

Figure 2 is an overview of the proposed traffic management system. (See also the literature for the detailed primary-backup replication used in this system (Hasebe et al., 2011; Hasebe et al., 2014)). As explained above, in our proposed traffic management system, one node is selected as the central server. The central server receives the travel demand sent by the passengers using smart phones and similar devices and periodically determines the optimized travel schedule with respect to the waiting times of the passengers and the traveling costs. Furthermore, this schedule is delivered to each node server and gives instructions to each vehicle arriving at a node. (The reason for passing through node servers is that this method is cheaper than constructing a communication network that issues direct commands from the central server to each vehicle in a wide-area transportation network.) Simultaneously with the distribution of the schedule, the central server periodically distributes information on the travel demand accumulated at the central server as an update state to the node servers.

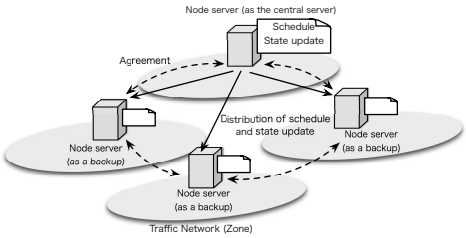

Figure 2: Overview of the traffic management system.

In addition to the above process, all node servers continually execute the optimistic consensus algorithm originally introduced in (Hasebe et al., 2011) in parallel to agree on the current central server. If the central server fails, according to the failure detector installed at each node server and the consensus algorithm, another node server will become the new central server. (In this sense, the central and node server are hereinafter referred to as the *primary* and *backup*, respectively.

4.2 Fault Tolerant Mechanism

The proposed fault tolerant mechanism in the traffic management system is realized by three processes.

1. Failure detection
2. Periodic execution of the optimistic consensus algorithm of (Hasebe et al., 2011)
3. Primary-backup communications

4.2.1 Failure Detection

Failure detectors are popular applications that are responsible for detecting server failures or crashes in a distributed system. Our proposed system is based

on the usual heartbeat-style technique; i.e., each node periodically transmits its own heartbeat packet while simultaneously monitoring the heartbeat packets received from other servers. If a heartbeat is not observed within a threshold time interval, the receiver considers the sender to have failed.

4.2.2 Optimistic Consensus Algorithm

The agreement algorithm decides which server is central among node servers. According to this algorithm, a value to be agreed upon is proposed by a server called the coordinator. The coordinator is in charge of a server with the lowest server ID among the servers judged to be normal. The coordinator proposes the role of each server, and each server agrees with the proposal. A proposed value is created so that any one of the servers judged to be normal can be set as the central server and other servers are usual node servers.

4.2.3 Primary-backup Communications

Primary backup communication replicates the state of the central server to other node servers. This communication also receives the request sent by the passenger, calculates the operation schedule, and returns the response. The state update is transmitted from the central server to the other servers. Here, the state update includes the passengers' requests that have been received so far, the latest operation schedule, and the schedules distributed so far to each node server. In addition, the latest schedule is delivered by this communication.

4.2.4 Behavior of Vehicles

Each lead vehicle informs the node server that it has arrived at the node and receives an instruction from the node server. In particular, when arriving at the node, if the node server is out of order, the lead vehicle runs on a predetermined emergency route while considering the destination of the present passenger in travelling to another node. The vehicle then receives the latest schedule and continues operation.

5 EXPERIMENTS WITH IMPLEMENTATION

To verify the correctness of our proposed mechanism and to demonstrate the availability with various types of server and network failures, we conducted exprements with our current prototype implementation. The modules described in the previous section behave independently and communicate with each other by means of asynchronous message passing. To implement these concurrent processes, our prototype was developed by Scala (Odersky et al., 2008). In particular, the interprocess communications were implemented by means of Akka, which is a Scala Actors library.

5.1 Experimental Setup

The experiments were conducted on 12 identical PC servers, each of which was equipped with Intel Xeon CPU E3-1220L V2 2.30 GHz, two HDDs with 1TB capacity, and 4GB of memory. The OS is Ubuntu 16.04.3 LTS. Each server was connected to a single switch via a 100Base-T network adapter. In our implementation, each node server, lead vehicle, and trailer was implemented as a single actor.

In this experiment, we assumed that there were 7 node servers, 2 lead vehicles, and 3 trailers in the transportation system. The failure detector transmitted the heartbeat every 10 seconds and confirmed the heartbeat received every 20 seconds. The agreement algorithm executed every 60 seconds and the upper limit time for the coordinator to wait for replies was 20 seconds. In addition, usual network delay was mimicked by generating artificial random delay up to 5 seconds for all communications.

As examples of possible failures in real systems, we considered the follwoing four cases.

Case 1. Failure of the majority of nodes: four node servers (Server 1–4), including the primary, fail at the same time.

Case 2. Network delay: communication delay occurs in the network between two groups of node servers, Servers 1–4 and 5–7, and then the system recovers.

Case 3. Network partitioning: network partioning occurs, splitting the node servers into two groups, Servers 1–4 and 5–7, and then the system recovers.

Case 4. Failure to receive operation schedule: a lead vehicle that arrived at a node fails to receive the operation schedule from the node server.

5.2 Experimental Results

5.2.1 Case 1: Failure of the Majority of Nodes

Fig. 3 illustrates the behavior of the system when the majority of node servers (Servers 1–4) fail and the system recovers after a lapse of 30 and 300 seconds, respectively.

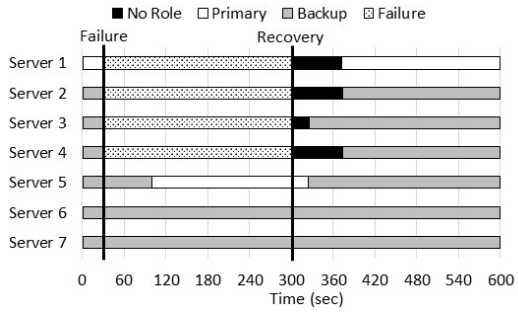

Figure 3: Case 1. failure of the majority of nodes.

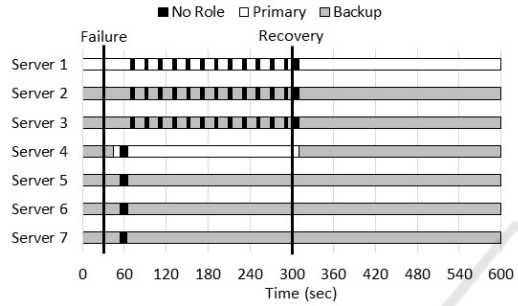

Figure 4: Case 2. network delay.

In this experiment, after 60 seconds from the failure, the consensus algorithm is executed for the first time after the failure. In this execution, the procedure of the consensus is continued even though the majority of node servers fail, and then, after 100 seconds from the failure, Servers 5–7 agree with the value that makes Server 5 primary and Servers 6 and 7 backups, and the role of central server was inherited. Furthermore, after 60 seconds from the recovery, the consensus algorithm is executed again in which all servers including Servers 1–4 participate. As a result, after 20 seconds, agreement is reached by the execution of the algorithm that makes Server 1 primary and the rest backups.

5.2.2 Case 2. Network Delay

Fig. 4 illustrates the behavior of the system when communication delay occurs in the network between two groups of node servers, Servers 1–4 (say, Group A) and Servers 5–7 (say, Group B), and then the system recovers after a lapse of 30 and 300 seconds, respectively.

As a result of executing the consensus algorithm immediately after the occurence of delay, agreement is reached with the value that makes the Server 1 primary, as before the delay. After 40 seconds, the consensus algorithm is executed independently in each of Groups A and B. As a result, in Group A, Server 1 becomes primary while Servers 2 and 3 become backups. On the other hand, in Group B, Server 4 becomes primary and Servers 5–7 become backups.

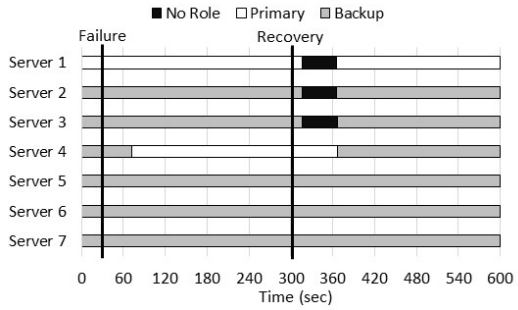

Figure 5: Case 3. network partitioning.

Furthermore, in the execution of the next consensus algorithm, since the decision value of the previous agreement in Group A has not arrived at Group B caused by the communication delay, the role of the servers in Group B is not given.

In the subsequent executions of the consensus algorithm, in Group A, the case where roles are not given because the previous agreed value in Group B arrives and the case where they agree on the suggested value of Server 1 alternately appear. On the other hand, in Group B, Server 4 becomes primary and Servers 5–7 continue to be backups.

As a result, in the period when the communication delay occurs, although the system configuration often changes, there is no time in which the primary does not exist. Also, after recovering from the communication delay, the system immediately returns to a normal state, where Server 1 becomes primary and the remaining servers become backup.

5.2.3 Case 3. Network Partitioning

Fig. 5 illustrates the behavior of the system when network partitioning occurs between two groups of node servers, Servers 1–4 (say, Group A) and Servers 5–7 (say, Group B) and then the system recovers after a lapse of 30 and 300 seconds, respectively.

After the occurrence of network partitioning, the consensus algorithm is executed independently in the two groups. As a result, Server 1 and Server 4 are chosen as primary. This suggests that although the consistency of the system cannot be guaranteed, it is possible to avoid the interruption of services due to the loss of the central server. When recovering from the network partitioning, after a lapse of 360 seconds, the consensus algorithm is executed by all servers, and after 20 seconds, they reach the agreement that returns the system to the state before the occurence of the failure.

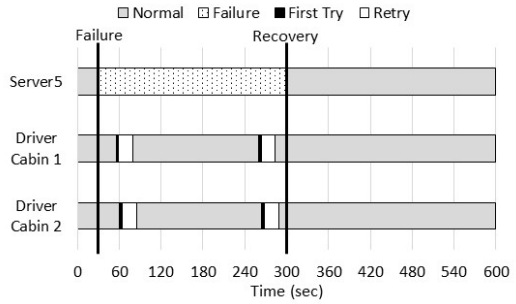

Figure 6: Case 4. failure to receive operation schedule.

5.2.4 Case 4. Failure to Receive Operation Schedule

Fig. 6 illustrates the behavior of the system when Lead vehicle 1 arrives at the node where the server 5 is located after a lapse of 55 seconds. In this experiment, Lead vehicle 1 tries to receive the operation schedule from the server in 4 seconds immediately after arrival, but could not receive it due to a failure of the server. Therefore, this process is repeated five times in succession. Still, because the vehicle could not receive the operation schedule, it moves to another certain node in order to try to receive the operation schedule from another server. As a result, it can be avoided that the vehicle keeps waiting under the failed server, and the operation of the vehicle can be continued.

In addition, Lead vehicle 1 arrives again at the same node after 259 seconds, but similarly it was unable to receive the operation schedule in the first 4 seconds, and repeatedly waited for reception five times and moved to another node again. Also, for the Lead vehicle 2, after arriving at the same node, it behaves similarly to the Lead vehicle 1 without terminating the transportation service.

6 CONCLUSIONS AND FUTURE WORK

We proposed a fault-tolerant mechanism for the traffic management system of a last-mile transportation service. The basis of our mechanism is to use a primary-backup replication technique. More precisely, one of the node servers is selected as the central server and the other servers periodically receive state-update messages, thereby allowing another node server to take over when the central server fails. In particular, the primary-backup replication used in this study is an optimistic version previously introduced (Hasebe et al., 2011). This makes it possible to tolerate various patterns of failures, including the simultaneous failure of the majority of servers and network partitioning.

We are currently conducting demonstration experiments using golf carts to realize this transportation system. In future work, we will investigate the transportation system further to refine our implementation in experiments.

REFERENCES

Defago, X. and Schiper, A. (2004). Semi-passive replication and lazy consensus. *Journal of Parallel and Distributed Systems*, 64(12):1380–1398.

Dijkstra, E. W. (1974). Self-stabilizing system in spite of distributed control. *Communication of the ACM*, 17(11):643–644.

Dolev, S. (2000). *Self-Stabilization*. MIT Press.

Dolev, S., Kat, R. I., and Schiller, E. M. (2010). When consensus meets self-stabilization. *Journal of Computer and System Science*, 76(8):884–900.

Hasebe, K., , Tsuji, M., and Kato, K. (2017a). Deadlock detection in the scheduling of last-mile transportation using model checking. In *15th IEEE International Conference on Dependable, Autonomic and Secure Computing*.

Hasebe, K., Kato, K., Abe, H., Akiya, R., and Kawamoto, M. (2017b). Traffic management for last-mile public transportation systems using autonomous vehicles. In *IEEE 3rd International Smart Cities Conference*.

Hasebe, K., Nishita, N., and Kato, K. (2014). Highly available primary-backup mechanism for internet services with optimistic consensus. In *IEEE Third International Workshop on Cloud Computing Interclouds, Multiclouds, Federations, and Interoperability*, pages 410–416.

Hasebe, K., Yamatozaki, K., Sugiki, A., and Kato, K. (2011). Self-stabilizing passive replication for internet service platforms. In *4th IFIP International Conference on New Technologies, Mobility and Security*.

Odersky, M., Spoon, L., , and Venners, B. (2008). *Programming in Scala*. Artima.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

Stodden, D. (2007). Semi-active workload replication and live migration with paravirtual machines. In *Xen Summit, Spring 2007*.