Doctoral Dissertation

A Constitutional Method for Massive and Complex Simulation Infrastructure

Akihito Kohiga

Main academic advisor Yoichi Shinoda

Japan Advanced Institute of Science and Technology
Graduate School of Information Science

Dec, 2020

Contents

1 Introduction **1**

1.1 Simulation Technology 1

1.1.1 Overview 1

1.1.2 Simulation Methodology 2

1.1.3 My Target Application 4

1.2 Massive and Complex Simulation Infrastracture 5

1.2.1 Requirements 5

1.2.2 Overview 7

1.2.3 Use Case 10

1.2.4 Advantage 12

1.3 The Main Problems to Realize a Massive Simulation Infrastructure 15

1.3.1 Software Architecture for Distributed Simulation 16

1.3.2 Deploy Method of Simulation Components 17

1.4 Research Objective and Summaries 17

1.4.1 Research Objective 17

1.4.2 Research Summaries of Each Part 18

A Software Architecture for Extensible and Composite Simulation . 18

A Deploy Mechanism using Voronoi Dividing for Scalable Geographical Simulation . . . 18
A Priority Control Method based on Predicting The Simulation Events 19
1.5 Dissertation Composition . . . 20

2 Related Works **21**
2.1 Network Simulation based on Virtual Machine . . . 21
2.2 Traffic Simulation Tools . . . 22
2.3 Wireless Emulation Tools . . . 22
2.4 Simulation Environment for Self-driving AI . . . 23

3 A Software Architecture for Extensible and Composite Simulation **26**
3.1 Overview . . . 26
3.2 Functionalities . . . 27
3.3 Important Concepts . . . 29
3.3.1 Independent Information from Simulator . . . 30
3.3.2 Simplify Connectivity and Concurrency Support . . . 34
3.3.3 Participation of Many Users and AIs . . . 36
3.4 Case Study . . . 38
3.4.1 Overview . . . 38
3.4.2 Input Stream (wheel and pedal) . . . 39
3.4.3 Output Stream (camera) . . . 40
3.4.4 Collision Detection (signaling) . . . 44
3.5 Implementation . . . 46
3.6 Preliminary Evaluation . . . 51
3.6.1 Three Evaluation Cases and Environment . . . 51

3.6.2 Result . . . 53
3.7 Discussion . . . 55
3.8 Conclusion of this chapter . . . 56

4 A Deploy Mechanism using Voronoi Dividing for Scalable Geographical Simulation 57

4.1 Problem . . . 58
4.2 Geographical Distance based Virtual Machine Deployment . . . 59
4.3 Voronoi Decomposition of Simulated Map . . . 60
4.4 Load Balancing . . . 62
4.4.1 Divide an Area Cluster, Add a Physical Machine . . . 62
4.4.2 Breakup an Area Cluster, Remove its Physical Machine . . . 63
4.4.3 Partial Exchange of Area Management . . . 64
4.4.4 Reorganize the Area Management . . . 66
4.5 Simulation Architecture . . . 68
4.6 Evaluation . . . 69
4.6.1 Evaluation Circumstances . . . 70
4.6.2 Evaluation . . . 71
Physical machine utilization . . . 71
Number of sessions between physical machines . . . 72
4.7 Discussion . . . 74
4.8 Conclusion of this chapter . . . 75

5 A Priority Control Method based on Predicting The Simulation Events 76

5.1 Problem . . . 76
5.2 Related Works . . . 78

5.2.1 Task Scheduling for Cloud Computing 79
5.2.2 Virtual Machine based Simulation Environment 80
5.3 Solution 81
5.3.1 Basic Concept 81
5.3.2 Definition of Interaction 83
5.3.3 Prediction of Interaction 86
5.3.4 Grouping 87
5.3.5 Control Flow (set high priority to each vehicle) 89
5.4 Evaluation 92
5.4.1 Evaluation Environment 92
5.4.2 Evaluation Points 96
5.4.3 Result 97
Fundamental evaluation 97
Influences on the increasing the number of vehicle 99
Influences on the increasing the duration of a simulation 100
Differences of the number of interactions between random walk and traffic jam 101
Differences of the number of interactions between several sensor scopes 102
5.5 Discussion 103
5.5.1 Evaluation Summary 103
5.5.2 Consider an Operation Under the Long Duration 105
5.5.3 Consider the Combined Usage of Voronoi Decomposition and Priority Setting 106
5.6 Conclusion of this chapter 106

6 The other topics for a massive distributed simulation 108

6.1 Time Synchronization 109

6.2 Connectivity to the Other Simulators 110

6.2.1 Weak Coupling Analysis and Strong Coupling Analysis 111

6.2.2 Individual cases of Cooperation 112

Global Environment, Climate and Disaster Simulation 112

Recognition, Behavior and Social System Simulation 113

Sound, Microwave and Peripheral Circumstance related Simulation 115

6.3 Domain Specific Cloud Management 116

6.4 Verification and Validation of computer simulation models 117

7 Conclusion 120

7.1 The Outcome of the Research 120

7.2 Social Significance 121

7.3 A Prospect of the Massive and Complex Simulation Research Field 122

Acknowledgement 123

Presented papers 125

References 126

List of Figures

1.1 Simulation methodology 3
1.2 Massive and composite simulation infrastructure overview 8
1.3 data flow and simulation components from information layer to user layer . 11
1.4 data flow and simulation components between information layer and user layer 12

3.1 Overview Design of My Simulation Environment 27
3.2 Ideal Relationship between Information and Simulator 31
3.3 A concept image of simulation network in my simulation environment . . . 35
3.4 An usage of wireless communication in a blind intersection. (a) function blocks in a wifi formatter. (b) A driver's view encounted a vehicle behind the wall 37
3.5 Overview of self-driving simulation environment 38
3.6 Input stream from pseudo wheel and pedal devices 40
3.7 Output stream for a pseudo camera device 43
3.8 Collision detection mechanism with inter simulation communicator 45
3.9 Block diagram of the basic implementation 48

3.10 Camera view (a) map information from the information storage with GIS format. The red circle represents the range that the camera simulator get as the peripheral information. (b) Rendering result of the geometry information by Unity 50
3.11 Evaluation 53

4.1 Local concentration of network sessions in the case of Round Robin based VM deployment 58
4.2 dividing the simulation map into several areas and assign them into each physical machine 60
4.3 voronoi dividing based on junction 61
4.4 partial exchange of area management 65
4.5 enclave 67
4.6 simulation architecture 69
4.7 The number of physical machines to be used to execute 500 VMs 72
4.8 Elapse of the number of network sessions on every seconds (executed 500 VMs) 73
4.9 The total number of network sessions 74

5.1 Problem 78
5.2 Basic concept 82
5.3 Vehicle's position on each time period 83
5.4 Vehicle's position at a time point and the scopes of sensors 84
5.5 interaction search and grouping 89
5.6 Priority Control Flow in my simulation environment 91
5.7 Basic sequence of Evaluation 93

5.8 Cloud simulator (OMNET++) 94
5.9 Fundadamental evaluation of my approach 98
5.10 Influences on the increasing the number of vehicle 100
5.11 Influences on the increasing the duration of a simulation 101
5.12 Differences of the number of interactions between random walk and traffic jam 102
5.13 Differences of the number of interactions between several sensor scopes . . 104

6.1 Example output of simulation property 119

List of Tables

3.1 evalustion environment 52

3.2 Evaluation summaries 52

4.1 Programs to be used for evaluation 70

4.2 Traffic simulation parameters 71

5.1 Traffic simulation parameters 96

5.2 Configuration of the fundamental evaluation 99

6.1 Technology fields that uses simulation technology 111

Chapter 1

Introduction

1.1 Simulation Technology

1.1.1 Overview

A simulation is defined as an imitation of a system or a process with an approximate model. A system means a partial aspect isolated from a natural phenomenon, a complex structure or an invisible behavior of a society. A simulation helps us to understand a certain circumstance which is difficult to reproduce on the real world.

The purpose of simulation is to obtain a prediction. A prediction tells us the future behavior of a target "system" and its characteristic. For example, in the case of weather forecasting (rainfall simulation), we can obtain a "prediction" of the probability of precipitation in one week duration from the current wind direction and rain cloud occurrence in a region. According to the precipitation probability, we can prepare the schedules of some outdoor activities when we go out or prepare the evacuation plan if it is a typhoon. In

the case of a combustion simulation of an engine, we can obtain "predictions" of its properties (torque, RPM, etc.) based on a quantity and speed of gasoline injection by using mathematical equations to investigate its internal combustion efficiency. By the engine simulation, we can set the amount and speed of gasoline injection that will optimize the performance of the engine, without using the actual engine.

Predictions obtained through simulations are indispensable to our lives and are expected to be used in a wide variety of fields in the future.

1.1.2 Simulation Methodology

Simulation methodology is classified on the figure 1.1 [51]. There are two major categories. One is "Physical Simulation" and another is "Logical simulation". Physical Simulation uses an actual object to construct a model to be simulated. Logical simulation mostly uses a computer, instead of an actual object. In the following chapters, I mainly deal with "Multi agent model" which is one of the discrete system simulation.

Multi agent model is a simulation model composed of multiple agents. The Multiple agents run their own program, but they are independent with each other. Agents sometimes have "relationships" with the others nearby. Eventually, these relationships become a "mass movement". An objective of a multi agent model simulation is to observe the mass movement as a macro behavior caused by the relationship of many agents. Evacuation from a building in a blaze and traffic congestion are the typical situations of the multi agent mode simulation. Each evacuee has individual decision and they run to an exit not to be involved in fire. At a narrow corridor, they make a confusion and some people would die. Observing the result of the simulation, a building architect can improve the building

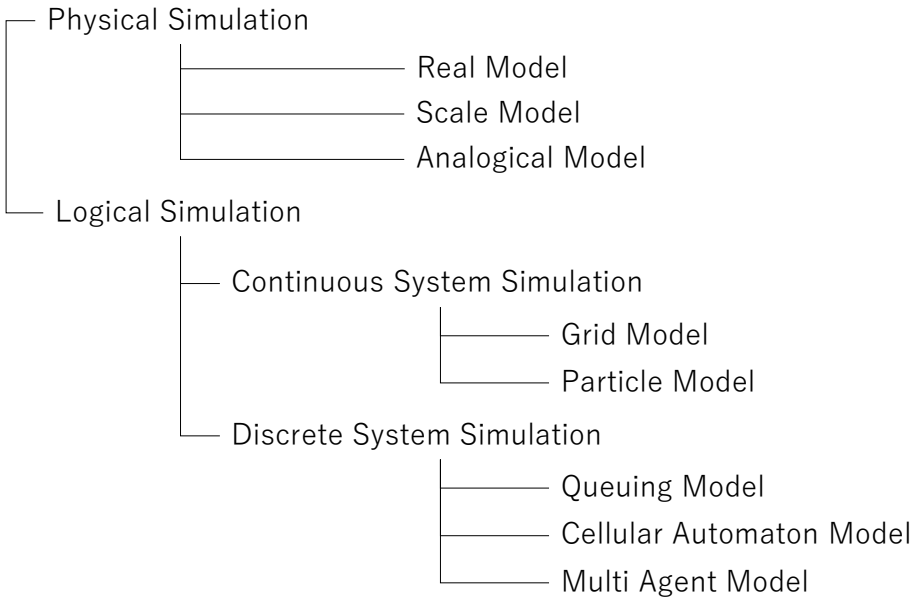

Figure 1.1: Simulation methodology

structure to lead a smooth evacuation or a building maintenance manager can make a great evacuation plan which remove a narrow route.

Our society is made up of individuals. Therefore, my simulation that represents social activities is essentially equal to agent-based models. There are many social phenomena, such as the transmission path of viruses and the relationship between economic activities and market activity, that have not yet been understood, and these problems can be represented by multi-agent models. My target application, shown in the next section, is also based on the multi-agent model.

1.1.3 My Target Application

I would like to establish a simulation environment as a testbed for automated driving technology, taking into account that mobility technology centering on automated driving will be especially important in the aging society with a low birthrate.

Mobility technology has received the most attention in recent years. The key technology of mobility technology is Artificial Intelligence (AI). AI is intelligence demonstrated by a computer. It takes over the human's "recognition", "inference", "natural language processing", "planning" and so on. An AI takes a long time to learn an appropriate behavior for each objective, therefore, many trials should be done before the product release.

Unfortunately, many serious accidents occur during the trials in these days. According to the website [3], 5 serious incidents occurred by self-driving vehicle are listed. Consequently, many precise computer simulations should be done before a field trial to prevent from serious accidents. Computer simulation can make a lot of situation which is too difficult and dangerous to run in reality. Computer simulation would reduce these serious accidents through the virtual trials. It is controversial that how many miles are enough to prove reliability of self driving technologies. No matter what metrics may be taken for proving its reliability, accidents in trials should be reduced as many as possible, and many simulation before actual trial increasingly become important we're convinced.

1.2 Massive and Complex Simulation Infrastracture

1.2.1 Requirements

Mobility technology consists of a variety of devices and programs. Camera images, LiDAR (Light Detection and Ranging) and accelerators are typical examples of these devices. An AI on a self-driving vehicle collects data from these devices, recognizes its surrounding circumstance, and determines the control amount of a vehicle. Therefore, my simulation environment needs the ability to generate data from various devices with various simulators.

The purpose of my simulation is to "improve the cognitive ability of AI" and "offer a test environment" for whole self-driving system. The accidents listed on the website [3] are caused by a lack of cognitive ability or by the circumstances that the AIs have never encountered during the preliminary tests. In other words, this suggests that there is a limit to how much accidents can be prevented by human-conceived test scenarios. Therefore, in my simulation environment, it is desirable to be able to reproduce a situation where some kind of unexpected trouble occurs as a result of the situation that a large number of vehicles run freely, instead of preparing a scenario in advance and conducting tests. In order to create such a contingency situation, it is possible to increase the probability of occurrence by running as many vehicles as possible to participate in the test.

In addition, when self-driving vehicles operate in the real world, several car companies will install their own independently developed AI in their self-driving vehicles. There is currently no experimental environment to test the situation when these different AI-equipped self-driving vehicles encounter each other on public roads. Testing multiple self-driving vehicles with different AI in the same space is meant to: 1. verify that they will

not collide due to misjudgment of each AI; 2. verify that better driving conditions can be achieved through self-driving vehicle's negotiation. The first means to verify that one vehicle's judgment does not influence another vehicle's judgment and create a dangerous situation when there are multiple free-running vehicles in the same simulation, and the second is to verify that multiple self-driving vehicles will not collide at an intersection with poor visibility. If the multiple self-driving AI can run on one simulation environment, we can test a negotiation protocol and it will allow the vehicles for smoother driving, rather than stopping in a hurry when an encounter occurs. Therefore, in my simulation environment, it would be desirable to be able to inter-test with several different self-driving AIs.

Based on the above discussion, the following three requirements are required for the simulation environment of the self-driving vehicle I am aiming for.

- The simulation environment must have an ability to give the both independency and interoperability to many kinds of simulators.
- The simulation environment must have an ability to execute and test multiple self-driving AIs simultaneously.
- The simulation environment must have an ability to induce unexpected traffic accidents without traffic scenario.

It is difficult to meet these requirements with existing agent simulators. As mentioned above, simulation is to simulate events by using a model that approximates the target "system". It has been a basic policy of simulation that the subspaces to be extracted as a system should be represented by a concise model as much as possible. The reason is that the more complicated the model is, the more difficult it is to analyze the causal relationship

between the model and the simulation results. According to this principle, conventional agent simulations are usually realized as a single application by simplifying the individual agents as much as possible. For example, artisoc [53] is a well-known agent simulation tool that recreates human interactions on a computer, such as the movement of a person at a certain location, but it is not a framework that is suitable for combining multiple models[1]. The same is true for other agent simulation tools.

Therefore, my objective is to construct an infrastructure that enables large-scale and complex simulations by connecting simulations that have been handled independently, such as those for human movement and wireless communication.

1.2.2 Overview

The objective is to construct a fundamental environment for large-scale complex simulations by connecting independent simulations such as human movement and wireless communication. The overview of my suggested simulation environments depicted in the figure 1.2.

The massive and complex simulation infrastructure environment is composed of five elements, including four layers and a middleware that mediates the transfer of information between them.

Simulation Layer is responsible for creating the data that composes a virtual spece. A virtual space includes buildings, pedestrians, the other artificial environment like radio wave and so on. Buildings are build on a virtual space by a spacial simulation tool. A pedestrian simulation or a traffic simulation will be executed by the other simulation tools.

[1]For example, if the information obtained by radio communication is used to select evacuation routes, the simulation of human movement and radio communication is required for each

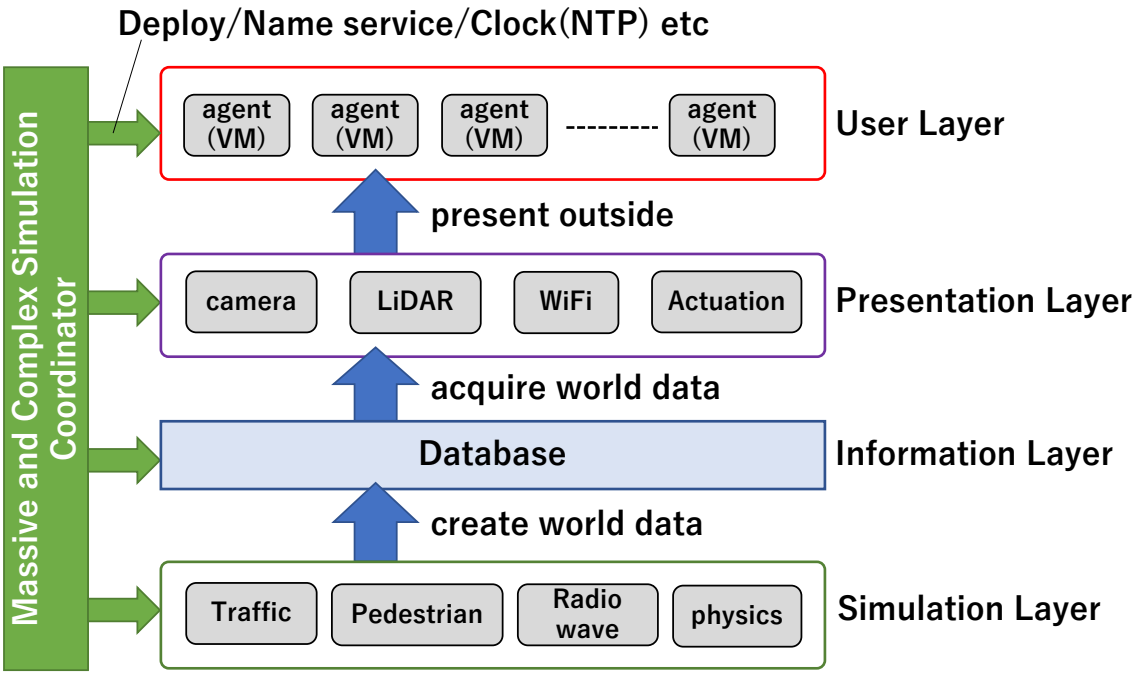

Figure 1.2: Massive and composite simulation infrastructure overview

They are made as the individual programs, do not exchange the data directory, but the results are stored on a same database in information layer. The data in the database will be shared with a data organizing program (I call it as **"a formatter"** later in this dissertation) in the presentation layer or be exchanged between the simulation tools are performed via the database.

Information Layer is responsible for storing all data. Information layer is composed of a database. In the case of a building data, the shape (width, height, depth), texture, and a number of windows will be stored. In the case of a vehicle, shape, velocity, acceleration and direction are the data of a vehicle. I call the data stored in a database as "world data" in the following sections. This database should also have generic APIs to store and extract the data and the database also should be scalable because it deals with many kinds of data which are created by many simulators.

Presentation Layer creates and offers the outside environment of the agents in User Layer. An formatter in this layer acquires the data from the database in the information layer and organizes data which a certain device would originally make. In the case of a camera emulator, it acquires the data of peripheral objects, makes 3D images, and sends them to the agent in the user layer which the camera device is attached.

User Layer is composed of many agent programs. In contrast to a traditional multi agent model simulation, the agents in my simulation environment are independent of the other agents. They are not included in a monolithic program but the independent applications. I am suppose to use virtual machine as an agent. The agents receive the outside environment data from the emulators which are in the presentation layer. Synchronization

of the simulation clock on each agent can be performed by the middleware I offer.

Massive and complex simulation coordinator is a middleware that mediates between multiple layers or components in each layer. This middleware prepares the database, deploys simulation components, and offers the name and clock service. A simulation component accesses the middleware to inquiry the place of another component. For example, if an agent needs WiFi emulation, this agent inqueries the place of the WiFi formatter that creates a packet drop rate (it depicted as a rectangle in the presentation layer), and make a connection directory between the agent and WiFi formatter.

Basically, my simulation environment is one of the multi agent model simulation because many agents are created on the user layer. Different points compared with the other multi agent simulation are the fact that my simulation environment creates the outside circumstances with individual simulation tools, coordinates the all components through middleware, shares the all data via database and all components appeared in my simulation environment are created independently by different developer.

1.2.3 Use Case

I describe a use case in the figure 1.3 and 1.4. This is a test environment of self-driving AI. There are three layers: Information layer, Presentation Layer, and User Layer. Information layer equal to a database. Presentation layer is composed of a camera, an actuation, and a LiDAR formatter. Agents in user layer is composed of virtual machines. These formatters create the inputs for the pseudo devices attached on a virtual machine. In the case of a camera formatter, it creates a view image from the world data (shapes of road, building, pedestrian and the other vehicles within a certain scope), and then sends the view image

to a pseudo camera device attached on a virtual machine. A self-driving AI on a virtual machine can see the outside view via the pseudo camera device.

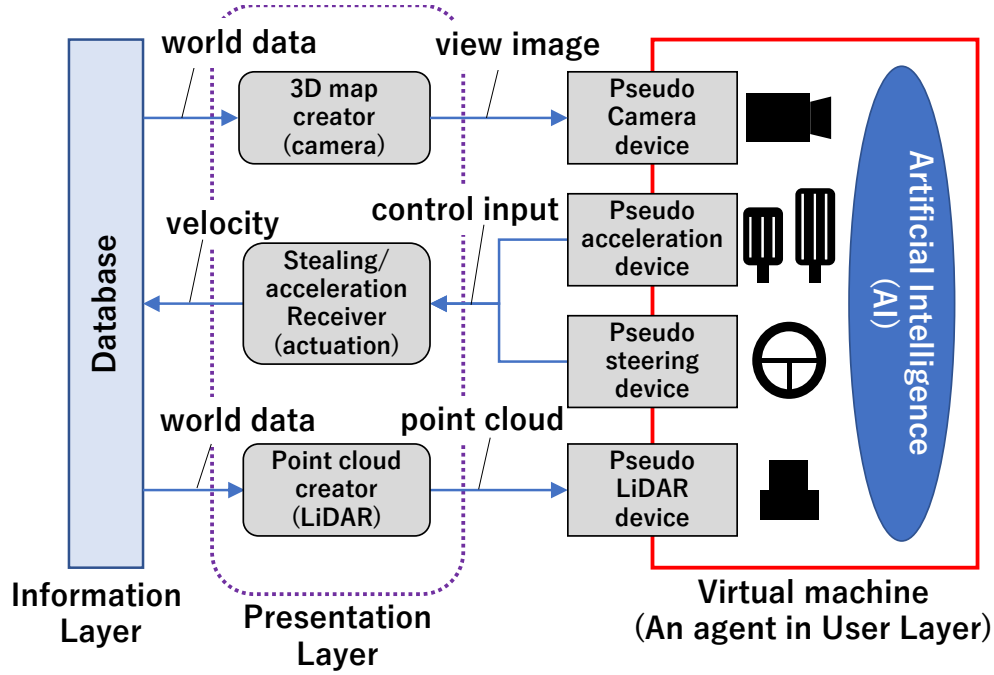

Figure 1.3: data flow and simulation components from information layer to user layer

Steering related components have the opposite direction of data flow. An actuation formatter receives control input (the amount of acceleration/brake pedal pressure, the angle of steering wheel) from the virtual machine and converts the control input to velocity, acceleration and direction of the vehicle. these outputs are stored on the database and will be used by simulators.

Meanwhile, simulators in simulation layer create the world data. There are three simulators in the figure 1.4 : traffic simulator, pedestrian simulator, radio wave simulator. Traffic simulator calculates each vehicle's position, velocity, acceleration and direction

from the object data on a virtual space. Pedestrian simulator also calculates the position of individual people. Radio wave simulator calculates the attenuation between each device. If two of them need to exchange the data, it will be done via database, so they do not have any direct connections. In other words, they are independent of the other simulators, except for database.

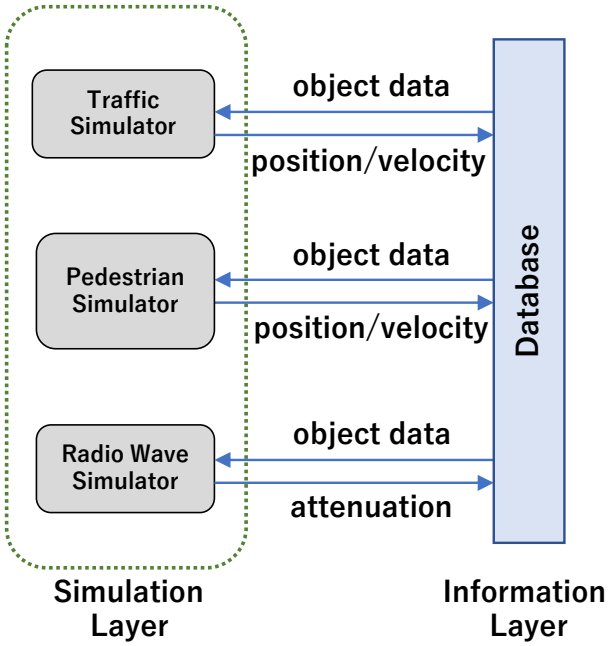

Figure 1.4: data flow and simulation components between information layer and user layer

1.2.4 Advantage

The following three items are the advantages of my simulation environment. I discuss these advantage in this subsection.

Independency of Simulation Components

Compared with the previous multi agent simulation, my environment has independency of simulation components. Simulation components are made as the individual programs, do not exchange the data directory, but the results are stored on a same database in information layer. I consider that independency of simulation components is significantly important for complex simulation like self-driving simulation. As I mentioned it in the previous section, many simulators make world data and self-driving AI receives the outside circumstance through many formatters. For making a simulator or formatter, special technical background is needed, so I assume the situation that many developers who have different technical background create their own simulators independently. Many previous simulation tools compel to use their special APIs to bind the simulation components made by different developer and it reduces the development speed. Therefore, independency of simulation components is significantly important, for making complex simulation with many developers.

Transparency for Application

My simulation environment is supposed to use a virtual machine as an agent in user layer. An agent can acquire the outside circumstance through the pseudo device drivers attached on the virtual machine. This means the fact that users of these virtual machines can execute their own application without any modifications and install whatever they may want. In the case of self-driving system, a user can export the inside environment of a virtual machine to an actual ECU (electronic control unit) on a vehicle. This approach is resembled to "Hardware-in-the-loop(HIL) simulation" [33]. HIL simulation is one of the real-time simulation, which enable to check an actual hardware module before assembling. My simulation environment, a virtual machine will be used, instead of an actual hardware.

HIL simulation is used for a strict hardware controller, for example, fuel injection on a vehicle. In contrast with HIL simulation, My simulation environment is supposed to be used for millisecond order application like AI on a self-driving vehicle. Both of them are effective for reducing development time.

Interoperability between Applications

Virtual machine technology is used as agent program, therefore, quite different self-driving AI can be executed on each virtual machine. They naturally understand the other agents existing nearby through the pseudo sensor devices. Therefore, on my simulation environment, many kinds of self-driving AI can interract each other without any simulation specialized programs. For example, I am going to think of a critical situation that two vehicles about to encounter each other at a blind corner. They must have communication to avoid a collision and the developers of two self-driving AI must implement negotiation protocol. The existing multi agent simulation tools comple to use special APIs to communicate with the other agents. In contrast with the existing simulation tools, my simulation environment will use common communication protocol like TCP, so the developers of self-driving AI do not need to implement any simulation specialized programs. This feature improves the interoperability between multiple self-driving AIs.

Interoperability between Simulation Components

My simulation environment enable to swap a part of the simulation components, and this feature is contribute to make a sophisticated and adaptive AI. We sometimes suffer from understanding what an AI think because of complicated parameters. In a case of a neural network, each synapse is composed of a set of parameter (weight and bias) and many synapses make a neural network. The parameters are fixed by a huge amount of input data,

and this is so called "learning". Recognizing handwritten character is a typical application of such neural network. Whether or not a handwritten character has a sort of "habit", a well-trained neural network recognizes the character correctly, but if it rarely fails to recognize, we can not find what was the trigger parameter for failure because there are many parameter sets in a neural network and it is impossible to check a parameter one by one. An object recognition in self-driving system is the same as recognizing of handwritten character. It is very difficult to find what cause the failure of the object recognition and the failure sometimes causes a traffic accident during self-driving test in the real world. My simulation environment enables to swap a part of the simulation components and this feature means the fact that my simulation environment change the input data to whatever we may want. For example if a self-driving AI failed to recognize a pedestrian in front of the vehicle, we can change the camera formatter to another one which has more higher resolution, or change the pedestrian simulator to another one which outputs the shape of well-detailed human. On the other hand, we can investigate a case, if a camera simulator is changed by another one which outputs a view with lower resolution, the self-driving AI can still recognize correctly. The ability of partial exchange of simulation components enable to make a variety of test cases and it leads to create a sophisticated and adaptive AI.

1.3 The Main Problems to Realize a Massive Simulation Infrastructure

There are three requirements of a simulation environment for self-driving system, which are mentioned on the section 1.2.

- The simulation environment must have an ability to give the both independency and

interoperability to many kinds of simulator and formatter.

- The simulation environment must have an ability to execute and test multiple self-driving AIs simultaneously.
- The simulation environment must have an ability to induce unexpected traffic accidents without traffic scenario.

I indicated that my simulation environment gave the answer for the first and the second requirement, giving the both independency and interoperability to many kinds of simulator and formatter, executing and testing multiple self-driving AIs simultaneously. However, the third requirement has not be solved yet, inducing unexpected traffic accidents without traffic scenario.

One approach to induce an unexpected traffic accident is to increase the probability of unexpected encounter of vehicles. I consider that a large amount of vehicles must run with their own individual decision on a large map to increase the unexpected encounter. In order to let a large amount of vehicles participate in one simulation, my simulation environment should be elastic and scalable.

I introduce the three major problems that prevent my suggested simulation environment from elastic and scalable. I will discuss the details in the following subsections.

1.3.1 Software Architecture for Distributed Simulation

A specific software architecture must be needed to implement the simulation environment I suggested. I depicted an overview in the figure 1.2 and indicated that there is a coordinator which coordinates communication between multiple layers, prepares a database to

store all data, and deploys every simulation components on a cloud. however, I have not indicated the individual functionalities yet. For example, an agent need the input data of a pseudo device, and an formatter serves the input data. A coordinator of my simulation environment must bind an formatter program and an agent without any dependency between them. I mentioned that a database stored all data in my simulation environment, but did not suggest the requirement of the database, I must research the requirement that the database should have.

1.3.2 Deploy Method of Simulation Components

I must consider the deploy algorithm of the simulation components. The simulation components like camera formatter or traffic simulator have quite different capabilities, so the computing power that they need also different. On the other hand, arbitrary two simulation components happened to have communication, especially such communication is happened on two or more than two agents. For example, communication for negotiation at a blind corner I mentioned is a typical case. I must research an appropriate deploy mechanism that satisfy computational and network bandwidth load balancing simultaneously. Otherwise, load imbalance occurs in a computing cloud and it degrades scalability of my simulation environment.

1.4 Research Objective and Summaries

1.4.1 Research Objective

My research objective is to create a massive and complex simulation infrastructure for self-driving AI testbed. I described the overview of the simulation infrastructure and

indicated the two major problems to embody it, those are an specific architecture and deploy mechanism. I would like to present the solutions for the two problems in this dissertation.

1.4.2 Research Summaries of Each Part

I would like to describe the following abstractions of three parts of my research.

1. A software architecture for extensible and composite simulation
2. A deploy mechanism using Voronoi dividing for scalable geographical simulation
3. A priority control method based on predicting the simulation events.

A Software Architecture for Extensible and Composite Simulation

A specific software architecture must be needed to implement the simulation environment I suggested. I designed and implemented a software architecture for a massive and complex simulation. A set of the software components is shown on this research and the data flow between the components is also described. I also introduced PostgreSQL database as a information layer because it has strong functionalities to store the geographical information. I implemented the three typical use cases: output stream, input stream, and signaling stream. I finally described the communication delay is enough small to assemble the whole simulation components.

A Deploy Mechanism using Voronoi Dividing for Scalable Geographical Simulation

Virtual machine(VM) based ad hoc network simulation executes VMs as moving objects (pedestrian, vehicle, etc) on which actual telecom applications and network routing pro-

tocols are executed. As the number of VMs increases, network congestion occurs because unknown network topology created by ad hoc network routing protocols makes a lot of network session over the physical machines. The network congestion limits the scalability of such simulation environment.

I suggest a deployment mechanism to reduce the network sessions over the physical machines. My mechanism executes VMs on a same physical machine, which are “geographically” close to each other. If the simulated position of the moving objects are changed, these VMs are also moved to another physical machine. My approach reduces the network sessions over the physical machines by about one-sixth in the case of 500 VMs and it contributes embodiment of a massive virtual machine based ad hoc network simulation.

A Priority Control Method based on Predicting The Simulation Events

As a simulation progresses with the above deploy mechanism, transient CPU overload sometimes occurs because of the spontaneous increase of VMs in an area. CPU overload makes a processing delay of a simulation component on a VM and the clock skew caused by a processing delay occurs between several VMs. The clock skew provokes inappropriate situation that two vehicles do not encounter at an appropriate place I expected to test a traffic accident and vice versa.

I suggest an interaction Based task group scheduling mechanism. There are several VMs which are executing self-driving AI we are going to verify the behavior, and also several VMs that interact with the target VMs to be verified. I anticipate these interactions before the self-driving simulation with a traffic simulator and set high priority to them. My approach reduces the processing delay at most one ninetieth in the case of 100 VMs executed on 9 physical machines.

1.5 Dissertation Composition

This dissertation is composed of the following chapters.

In the chapter 2, I describe the related works. This chapter is composed of the previous researches on multi agent simulation, the other simulation environments, virtual machine based network simulation frameworks, and simulation tools for self-driving AI. In the chapter 3, I describe the design and implementation of my simulation environment. In the chapter 4, I describe a deploy mechanism using Voronoi dividing for scalable geographical simulation. In the chapter 5, I describe a task group scheduling mechanism which is based on predicting the interactions during simulation. In the chapter 6, I mention the other topics related to massive and complex simulation environment. In the chapter 7, I conclude the above researches in my dissertation.

Chapter 2

Related Works

2.1 Network Simulation based on Virtual Machine

Mininet [29], CORE [8], IMUNES [58], Netkit [43], Cloonix [54] are the network simulation frameworks based on virtual machine technique. Basically, these frameworks consist of virtual machine as network nodes, virtual switch and router, GUI for configuration of network topology. An network interface device is basically connected to virtual switch and router and a real network packet is transferred to a destination through the virtual switch and router. These above frameworks can also connect to a real network. In addition to the connectivity to a real network, an actual network application can be run on a virtual machine in these simulation frameworks without any modification and this feature reduce the development cost of an network application. These frameworks also become a demonstration framework which gives the stakeholders strong impression. FEP [59], NEmu [11] are also network simulation frameworks but they focus on making wireless network.

None of them however supports ad hoc network simulation because the purpose of these frameworks are basically to make fixed network topology and give a test environment of

an network application on a fixed size of simulation environment. CORE, FEP, NEmu support wireless simulation to which user directory input connectivity scenario but do not calculate packet drop rate in accordance with vehicle's distance or some other barriers. The scalability of these frameworks are also ambiguous.

2.2 Traffic Simulation Tools

There are many traffic simulators for various purpose and they are basically divided into two categories. One is "wide area" traffic simulation. It is for observing overall traffic characteristic (ex. traffic jam). Another is "microscopic" traffic simulation. It is for demonstration of user experience and investigating feature of an individual vehicle (amount of exhaust gas in a certain circumstance, etc). SUMO [34] is one of the most famous "wide area" traffic simulator. SUMO supports various topics (ex. Different right-of-way rules, traffic lights, Different vehicle types, Multi-lane streets with lane changing, vehicle crash and so on). In contrast, An Open-Source Microscopic Traffic Simulator [52] and VISSIM [19] [44] focus on several pinpoint simulation (ex. traffic jam at a certain ramp joint, visualize a camera view mounted on a vehicle, , IBM Mega Traffic Simulator [40] can create a gigantic traffic simulation by utilizing distributed computing architecture. I need the vehicle's position at wide area traffic, therefore, I chose SUMO simulator for generating vehicles position.

2.3 Wireless Emulation Tools

ORBIT [46], Qomet [14] are wireless emulation tools. These tools emulate packet drop rate (or radio attenuation) in accordance with distance between vehicles and barriers (buildings,

wall, etc), so that they also have simulation map and simulate vehicles movement, in addition to calculating packet drop rate. The packet drop rate is set to an actual network interface device driver on a computer. These tools also work for test tools of an network application. or test for an network protocol.

2.4 Simulation Environment for Self-driving AI

There are many simulators for robotics and they are close to what I want to embody. V-REP [47] and online simulation architecture based on V-REP [42] (I call it Online V-REP) are suggestive architectures for making a composite simulation. V-REP is a robot simulation architecture. It mainly consists of run-time (Main client application), libraries (V-REP engine), simulation programs (main, child, callback scripts), plugins (to connect clients like ROS). The run-time also loads 3D models and a scene (virtual space). Online V-REP is an online version to run the simulation programs online. V-REP is matured for a precise robotics simulation, but it does not have enough scalability yet, same as the other simulators like Unity [6], Gazebo [7], and so on.

There are also some simulators designated to self-driving technology, Carla [18], Udacity's Self-Driving car Simulator [5] are the two of famous driving simulators which can attach an user program. They are based on Unity and several parts (ex. Lidar, camera, etc) are added.

Considering my ideal simulation environment on which developers can create a simulator independently and users can choose and assemble whatever they want to simulate, these following points would be problems of the above simulators.

- Special formats and APIs to store data.

- Direct connection of each simulation programs
- Single user basically uses the simulation environment.

First, It would be one of the problems that they have special formats and APIs to store data. V-REP (and the other simulation environments) has a special format to store a scene and 3D object (ttt file) and APIs to access inside it. This point would be a bottleneck, if many simulators access this file simultaneously. On the other hands, V-REP maintainer must create APIs whenever it may be needed. It is also a burden for a simulation environment developer. Therefore, generic and popular storage system like a relational database is appropriate. It has a long time knowledge and experience how to make the database bigger and keep good throughput.

Second, It would be one of the problems that simulation programs are directly connected with each other. V-REP executes a simulation program (called main and child scripts) as a thread or co-routine. And then, they are controlled by V-REP scheduler, which program follows the other programs. This type of execution is similar to the other environments. The thread execution is basically closed inside single computer. In the point of scalability, it would be better that a simulation program are executed as a relocatable process on several computers.

Third, It would be one of the problems that only single user can use the simulation environment. V-REP (and the other simulation environments) does not have any particular mechanism to participate in many users in the simulation environment. In another word, it is quite difficult to make an "interactive simulation" on the existing simulators. Interactive simulation indicates that several users and AIs can participate in one simulation environment and interact each other to make a certain situation. In the future motorized

society, it is obvious that there are huge amount of vehicles on various roads, and they are the mixture of the human controlled and self-driving vehicles and self-driving vehicles are controlled by the different AIs created by the different companies. We're convinced that this mixture makes unanticipated situation and the previous accidents listed on [3] are the example situations created by human and self-driving vehicles interaction.

Chapter 3

A Software Architecture for Extensible and Composite Simulation

3.1 Overview

I describe the overview design of my simulation environment in Figure 3.1. The overview consists of the extensible and composite simulation environment, simulation users and simulation program developers. Simulation users has their own computer and several pseudo device (kernel module) are attached on it. The extensible and composite simulation environment consists of simulators (vehicle, drone, camera, collision, pedestrian in this figure), several functionalities that I provide, and basic system software (cloud OS, RDMS, SQL). Simulation users send their input and receive the output from simulation environment via a pseudo device. Simulation program developers can make their simulator with minimum dependency from the other simulators. My simulation environment coordinate a simulation which an user want to simulate in accordance with a scenario file. Specifically, the simulation environment instantiate the simulators at first, and then connect the simula-

tors to user's pseudo device via I/O manager and start simulation. All of the information in a simulation are stored in the relational database management system (RDMS). Each simulator can access the information storage whenever it may be needed. each simulator is executed on a computer in a cloud. In the following subsection, I would like to explain the functionalities of my environment.

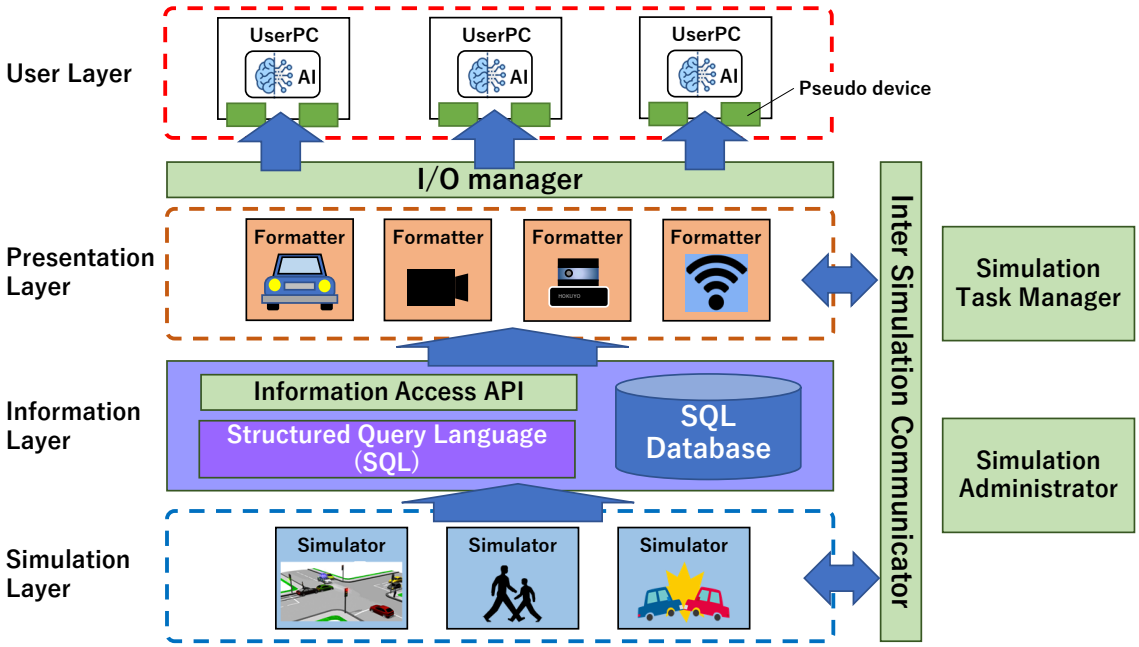

Figure 3.1: Overview Design of My Simulation Environment

3.2 Functionalities

I would like to explain the functionalities of my environment in this subsection. An actual use-case will be discussed in the later section.

Simulation Administrator Simulation Administrator receipts a user's request with a simulation scenario at the beginning of a simulation. a simulation scenario at least consists of user's profile (IP, port numbers to receive for I/O), simulation related information which simulator should be run and relationship information between simulators, map information and so on. Simulation Administrator takes over the instantiation process of each simulator to Simulation Task Manager and it also registers user's profile to I/O Manager to bind the simulators and the pseudo devices in the user's computer later.

Simulation Task Manager Simulation Task Manager executes several simulators in accordance with a simulation scenario and connects the simulators to the pseudo devices in a user's computer. Specifically, Simulation Task Manager registers the simulator related information to I/O Manager and then, I/O Manager makes the connections between the simulators and user's computer. The simulators will be instantiated as a virtual machine or a container.

I/O Manager I/O Manager makes several connections between a user's computer and the simulators. I/O Manager receives user's profile from Simulation Administrator at first, and then waits until Simulation Task Manager instantiates the simulators and send the simulator related information. After that, I/O Manager binds the simulators and user's computer. Actual binding process is to forward the simulators output to the pseudo device in the user's computer and vise versa. I/O Manager behaves like a router on the Internet and it conceals the place of the simulators and the user's computer each other because of "location transparency".

Inter Simulation Communicator Inter Simulation Communicator connects the simulators each other. This functionality is resembled to I/O Manager in the point of administrating connectivity, but Inter Simulation Communicator uses alternative ways to make the connections. It uses the networking if there are two simulators located on the different computers on a cloud and it also uses the shared memory if they locate in a same computer because networking has much heavier than the shared memory mechanism because of layered processing.

Information Access API Information Access API is a library to get the simulation related information. SQL has a basic role to get the information and Information Access API simplify the way to access it. If there are no API to get an appropriate information, user can easily customize the APIs. On the other hands, SQL engineer can insert a sophisticated API or a set of hardware and APIs. It will improve the performance to access the RDMS.

3.3 Important Concepts

The objective of my architecture is “many simple simulators make a massive composite simulation”. In the discussion in PROBLEMS chapter, it is very hard to make composite simulator for drone or self-driving system as one application because of its complexity. Each simulation needs each domain knowledge. For example, if we want to make a window simulator for drone, we need the knowledge of aerodynamics, if we want to make a traffic simulator, we need the knowledge of queuing theory. Therefore, it becomes more natural conclusion that each domain engineer makes each simulator and makes it composite

whatever it may be needed in the case of each simulation.

My objective “many simple simulators make a massive composite simulation” is parsed to the following three concepts.

- Independent information from simulator
- Simplify connectivity and concurrency support between simulators as much as possible
- Participation of many users and AIs

I would like to discuss the three concepts in the following sections.

3.3.1 Independent Information from Simulator

Each simulator should be executed independently from the other simulators. In order to do that, complete separation of information from simulator is the most important factor.

In many virtual space simulators like Gazebo, Unity, Carla simulator, etc, we must made 3D maps and buildings first as a basement, then put vehicles, pedestrians, and the other obstacles. In addition to that, camera, LiDAR, and the other sensors will be attached on each vehicle. These kinds of “monolithic” simulators imply several unsophisticated points, related to “scalability”. For example, LiDAR simulator don't need 3D map (rendering), but just needs “numerical information” of peripheral area, a walking simulator with pedestrian on a map don't need 3D map, but just needs 2D map. Too much information needs much processing power and that degrades capital expenditure (CAPEX) for computer simulation.

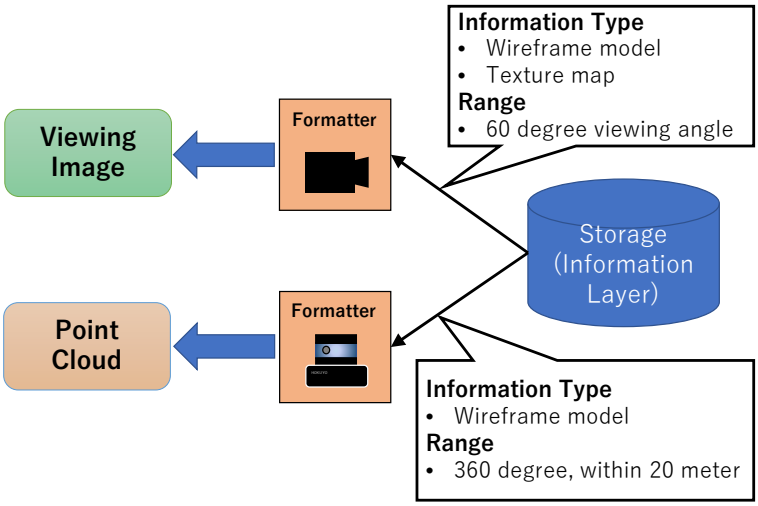

Figure 3.2: Ideal Relationship between Information and Simulator

Figure 3.2 describes a processing flow from storage to the formatters. In the figure, two formatters acquire the difference types of informations from the storage. In the case that two formatters acquire the peripheral building information, a LiDAR formatter just needs the wireframe model of buildings as the peripheral building information. In comparison of LiDAR formatter, a camera formatter needs the both the wireframe model and texture map of the buildings because the camera formatter must visualize the peripheral buildings. Instead of the camera formatter, LiDAR outputs a point cloud which does not include the building's surface texture and color information. On the other hands, a LiDAR needs the building information that located at all around of the vehicle, but the camera formatter just needs the building information that is located at a certaine degree viewing angle in front of the camera device. The most important thing is that the original data of the building is unique and both formatters acuire the part of the building information. If the both formatters have their own buiding infromation independently, it might be difficult to keep the consistency of the building information.

To suumerize the above discussion, I think that this principle includes three advantages and one disadvantage.

Advantage.1 Separation of information and simulator increases the choices of information storage and it improves scalability of the environment. If the information storage is a file and it binds to a simulation environment, we can not chose an appropriate information storage and it degrades the scalability of the environment. For example, RDBMS has good history of researches. If it is chosen for information storage of the simulation environment, we could receive several benefits from the previous researches of RDBMS. There are several SQL performance tuning (query optimizer) methods increase the throughput of database, and also using a hardware like GPU [12], FPGA [16] supports the rapid throughput improvement. Most powerful methods which RDBMS has would be geographical index search algorithm. In a drone or self-driving simulation, we will often use a geographical information to get the peripheral information around the target. For example, LiDAR simulator must make a dot cloud from the information of surround the vehicle (building, road, street tree, pedestrian, etc). R-tree [27] is an algorithm for making geographical index. R-tree stores the location information of each object as a tree structure. R-tree reduces search order to $\log_M n$ (M denotes maximum number of objects in each page). Q+Rtree [57] is also a geographical indexing algorithm for movable objects.

Advantage.2 Separation of information and simulator creates an option to choose information. The previous simulators like gazebo, Carla simulator, polygon is the only format to represent the shape of an object. Probably, using polygon format is appropriate for robot simulation which has very complex shape robots and complex behavior. For

drone or self-driving simulation, a precise 3D model is not always needed, rather light weight data is need because when an engineer want to create a prototype AI for avoiding several moving objects, all they need for a decision making is rough shape, location, and direction of the peripheral objects. of the obstacles in front of the target. OctoMap[32] would be a good example for the above purpose. It regard an object as "occupation of a certain cube" and OctoMap is enable to change the granularity of an object's shape through the change of a certain cube size. In a 2D picture, bitmap is similar idea as OctoMap. Interleaving OctoMap between simulator and information storage, an object could have many kinds of granularity of shape and it helps for developments of self-driving , drone or the other types of AI for unmanned vehicle.

Advantage.3 Separation of information and simulator accelerates studies of "shape of information" for future mobility. Self-driving technology must need precise map and the other information (ex. slope and dent of roads, construction, traffic jams, vehicles location, pedestrian, etc). There are several precise maps for self-driving technology. Geographic Data Files (GDF) 5.1 defined by ISO/TC204 Sub Working group [2] is discussing how express the road related information for self-driving system. Daimler [36], Here [31], Navinfo [24], University [50] and the other organizations is also struggling to make a precise map (it is called High Definition Map : HD map). However, almost all organization has not defined a format of "Dynamic information" yet, although they are going to fix "Static information". Dynamic information indicates information which changes its value as time passes (ex. construction, traffic jams, the other vehicles location, pedestrian, etc).

I guess the reason why none of them defines a format of dynamic information and it could be one of the answers that they could not clearly imagine "how the dynamic informa-

tion are used". Self-driving vehicle technology is under development and experiment phase and dynamic information is captured from the sensors on the vehicle and stored locally. To store these information would be enough during development and experiment phase, but there are several critical situation which should share the information each other. For example, taking avoidance behavior against the children who run out from a blind spot, entering a highway from a dead angle (the self-driving vehicle should know which space the vehicle is able to enter without seeing). In these case, shared dynamic information must be needed and standard format for dynamic information should be defined. From the above discussion, It is obvious that implementation of the dynamic information in the real world takes a long time. Therefore, repeatedly creating and refining a dynamic information in a simulation will fix an appropriate and sophisticated shape of it and I think that these trial accelerates the emergence of self-driving society.

Disadvantage.1 Separation of information and simulator creates time delay when simulator get information from storage. Time delay depends on the network distance between simulator and information storage. I evaluate the delay that my simulator makes later in this paper.

3.3.2 Simplify Connectivity and Concurrency Support

My simulation environment is categorized to one of the discrete event simulations. Discrete event simulation is also called agent simulation. Agent simulation is a type of simulation for watching behavior and relationship of each independent agents. Thomas C. Schelling creates one of the earliest agent simulation [49]. It proved the fact that people who has similar properties (sex, age, income, language, etc) eventually makes some groups around them even if they are placed in a random order at first. This simulation is not executed on a computer, but it embodies a basic concept of agent simulation. After that, agent

simulation spread its usage to an optimization of supply chain, logistics, social network, traffic jam forecast, and so on. For these agent simulation, many frameworks are developed. MASON [35] swarm [39] artisoc [53] are the examples.

Most important factor for improving connectivity and concurrency in a mobility simulation is "location transparency" of each simulators. It is an ability to connect each simulator to another without knowing where it is. Basically, a generic system software or middleware supports location transparency to increase scalability of the system and most powerful framework to supply location transparency is the Internet. Similar to the relationship between the Internet and their protocols, we should prepare an infrastructure as a simulation framework (Layer 3 protocols (ether, IP), DNS, router, socket API on the Internet), but we don't need to care communication related protocols (TCP, HTTP, SDP, etc). An appropriate protocol should be created or chosen between the simulators (Figure 3.3).

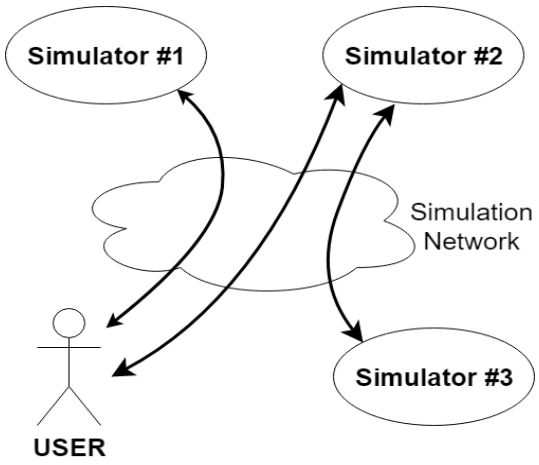

Figure 3.3: A concept image of simulation network in my simulation environment

3.3.3 Participation of Many Users and AIs

For creating a practical self-driving AI, it is the most important to experiment in a situation that many kinds of AIs and human controlled vehicle are running on one simulation environment. Discussed on the previous section, there are some critical situations when some AIs or human in the critical situation have to share their information (speed, position, and so on). They also have to negotiate each other to avoid the critical situation. For example, collision avoidance in a corner is a famous example of such critical situation. Two vehicles are about to enter an intersection, and come into collision. They have to negotiate who enters the intersection first. Literature [28] and [15] are two of the algorithms for collision avoidance. They control their speed and direction, accordance with another vehicle's behavior. They assumes that communication between them has no packet loss or noisy information. Especially, in the literature [15], it is referred that negotiation in a circumstance which has packet loss or noisy information is a future work. However, this kind of circumstance should be considered before an actual road test, otherwise unanticipated accident would be happen.

Figure 3.4 is an usage of wireless communication in my simulation environment. left side of this figure describes function blocks when two vehicles negotiate each other to avoid collision. right side of this figure describes driver's view encountered a vehicle behind the wall. My simulation environment also simulate this driver's view with the centralized information storage. A camera formatter get the peripheral information and make a screenshot of the view and send it to a pseudo device on an user's computer. The wireless simulator in this figure gets vehicles position and map information first, and then calculates the wireless network configuration(delay, packet loss late, bandwidth, etc) of each pseudo network devices on the user's computers. Finally the wireless simulator sets the wireless

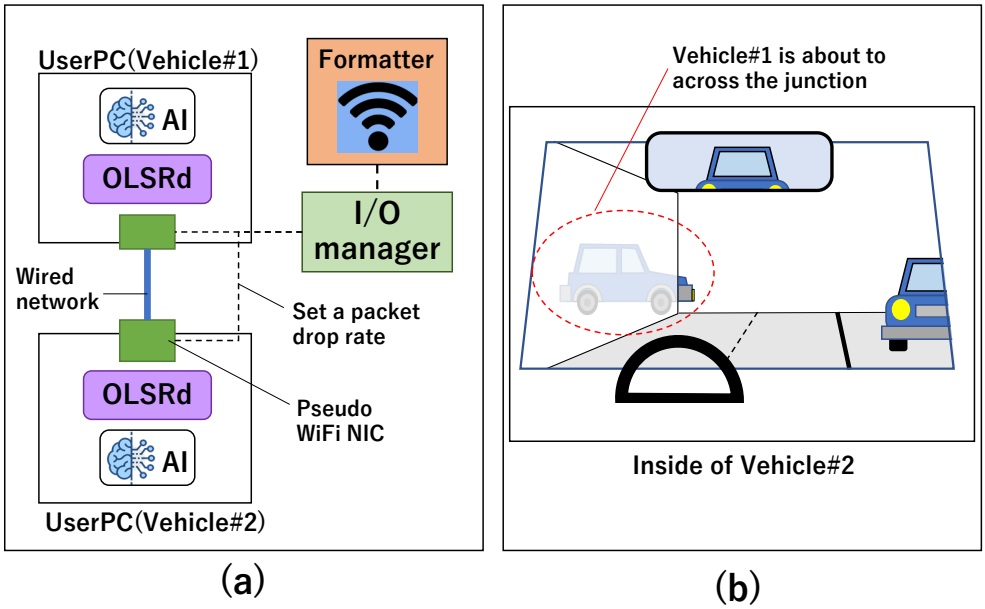

Figure 3.4: An usage of wireless communication in a blind intersection. (a) function blocks in a wifi formatter. (b) A driver's view encounted a vehicle behind the wall

network configuration to each pseudo network devices. After setting the wireless network configuration, an user's computer can communicate a certain user's computer with limited area where the radio wave from the pseudo device in the user's computer reaches.

In this example, I assume that users of each AI make direct connection with the other vehicles to negotiate, using ad hoc network mechanism. An ad hoc network routing algorithm (OLSRd in this figure) finds the peripheral vehicle and then make the direct connection. After the connection established, AI starts the negotiation to the other AI. The communication in this negotiation will have packet loss and noisy information, therefore, my simulation environment can make an actual situation about wireless environment, compared with the literature [15]. I think that my simulation environment can create more sophisticate collision avoidance algorithm with noise tolerance.

3.4 Case Study

3.4.1 Overview

I describe a basic set of the self-driving simulation in Figure 3.5. This self-driving simulation just receives camera view as input and controls the wheel and pedal as output. Some users use visual SLAM for mapping around the vehicle, make the direction and path with some algorithms (ex. A-star) and calculate the position, speed, direction of the vehicle. And the other users would use the different mechanism to implement self-driving mechanism. My simulation environment don't care what the users run in their computer. Besides the camera device, if the other sensors (LiDAR, sonar and so on) are needed, user can add simulators of the other sensors whatever it may be needed.

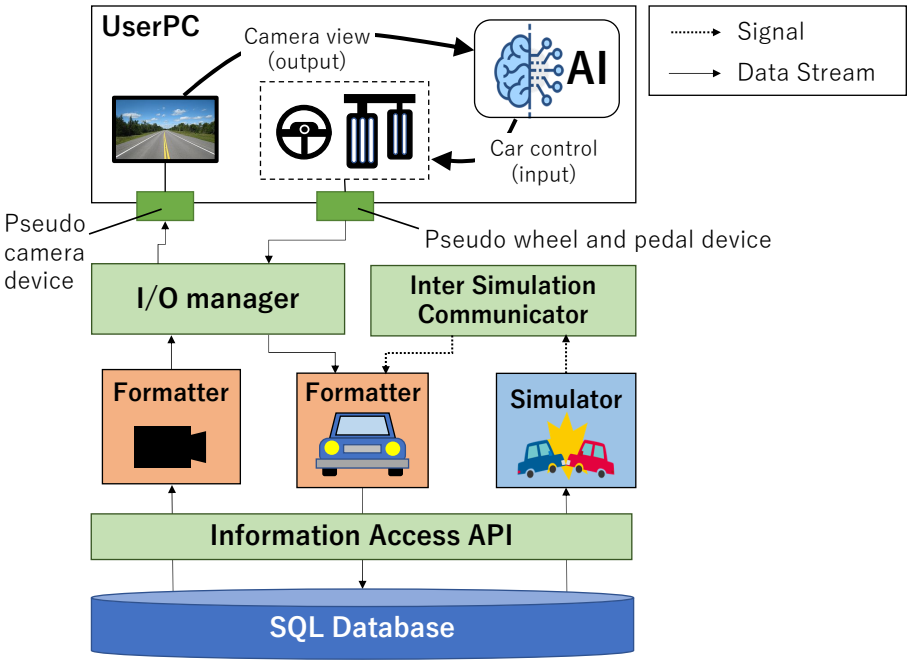

Figure 3.5: Overview of self-driving simulation environment

There are one formatter and two simulators, camera, vehicle and collision. Camera formatter output camera view to a pseudo camera device in an user's computer. Vehicle simulator receives user's inputs as manipulated variables of wheel and pedals. Vehicle simulator then calculates the speed, direction, position from the user's input and send the calculated values to the database. Collision simulator investigate collision every time with a certain interval and send a signal to the vehicles which made collision. Collision simulator also uses all of the vehicles position, direction and speed information. detection process is executed for each combination of the vehicles.

I explain the input, output and communication between simulators in the following sections.

3.4.2 Input Stream (wheel and pedal)

A vehicle simulator receives the manipulated variables of the vehicle and calculates the next position, speed and direction. Figure 3.6 describes the input stream of this simulation. The vehicle simulator also retrieves the current position, direction and speed information from the information storage. Algorithm 1 is a pseudo code of the vehicle simulator. There are basically three parts. One is the statement to get information. Another is the statement to calculate the next position, speed and direction of the vehicle (omit the description of the next speed and direction in this algorithm). The other is the statement to set the results in the information storage. The Get related functions are offered in my environment and the I/O related functions are also prepared in my environment. Therefore, a developer of the vehicle simulator can focus on to make what they really want to make (CreateNextPosition, etc). The developer of this vehicle simulator also don't need to care about how these parameter is used by a camera formatter. Calculation of the next position, direction and speed would also need some other parameters related to engine, friction of wheels and so

on. In this case, a developer can add such additional parameter to the information storage if necessary.

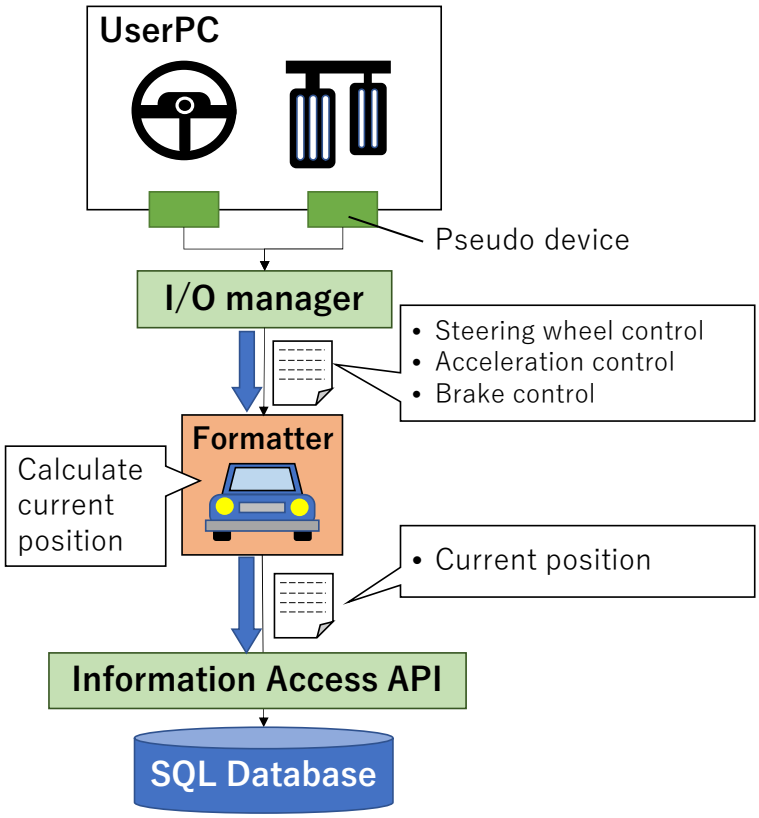

Figure 3.6: Input stream from pseudo wheel and pedal devices

3.4.3 Output Stream (camera)

Camera formatter output camera view to a pseudo camera device in an user's computer (ex. it can be seen /dev/video* in Linux). Figure 3.7 describes the data stream for output of a pseudo camera device. The camera simulator gets the peripheral information from the database at first, then makes 3D objects (the other vehicle, road, buildings) in a virtual space, finally makes screenshot every time with a certain interval and send them to a

Algorithm 1 Input the controls, make the next position

Require: id : $identification\ of\ this\ input\ stream$

```
 1: in ⇐ new IOManager.connect(id)
 2: while not end of the simulation do
 3:     position  ⇐ GetPosition(id)
 4:     speed     ⇐ GetSpeed(id)
 5:     direction ⇐ GetDirection(id)
 6:     steering  ⇐ in.GetInfo(id,"steering")
 7:     accel     ⇐ in.GetInfo(id,"acceleration")
 8:     brake     ⇐ in.GetInfo(id,"brake")

    (omit to create next speed and direction)
 9:     next_position ⇐ CreateNextPosition(
10:         position, (current position of the vehicle)
11:         speed,    (current speed of the vehicle)
12:         direction,(current direction of the vehicle)
13:         steering, (steering angle of the vehicle)
14:         accel,    (acceleration of the vehicle)
15:         brake,    (negative acceleration)
16:         )
17:     SetPosition(id, next_position)
    (and sleep during a certain interval)
18: end while
```

pseudo camera device of user's computer. Consequently, the user can receive the camera view by any application.

I also describe an example pseudo code of the camera formatter in Algorithm 2. There are three parts. One is the statement to get information, another is the statement to create screen image and the other is the statement to send it to a pseudo device on an users computer. My simulation environment offers IOManager related functions (line 1 and 13), Get related functions (GetPosition, GetBuidingInfo and so on, line 3 - 6). A developer of this camera formatter can also create a new Get related function with SQL. because the Get related functions simply replace the function call to SQL query, send it to information database and receive an appropriate information.

Algorithm 2 Output Camera View

Require: id : $identification\ of\ this\ output\ stream$
Require: $radius$: $range\ to\ get\ infromation$

```
 1: out ⇐ new IOManager.connect(id)
 2: while not end of the simulation do
 3:     position    ⇐ GetPosition(id)
 4:     buildings   ⇐ GetBuildingInfo(pos, radius)
 5:     pedestrians ⇐ GetPdestrianInfo(pos, radius)
 6:     vehicles    ⇐ GetVehicleInfo(pos, radius)

 7:     screen ⇐ CreateScreenImage(
 8:         position,    (position of camera itself)
 9:         buildings,   (position, shape of buildings)
10:         pedestrians,(position of pedestrians)
11:         vehicles     (position of vehicles)
12:         )
13:     out.send(screen)
    (and sleep during a certain interval)
14: end while
```

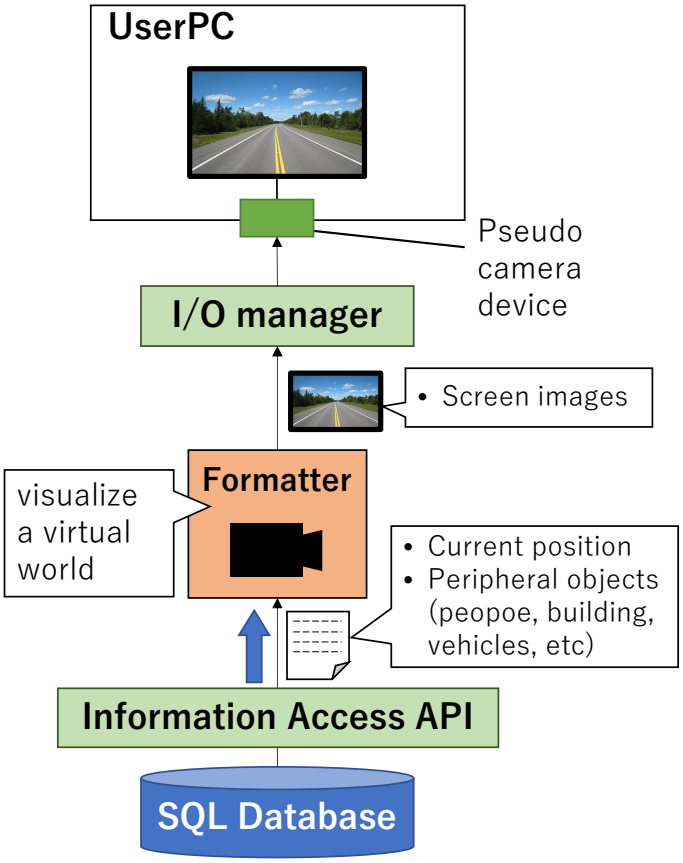

Figure 3.7: Output stream for a pseudo camera device

3.4.4 Collision Detection (signaling)

Collision detection simulator continuously try to find a collision between two vehicle. Figure 3.8 describes the mechanism to inform the collision signal to the corresponded vehicles. Collision detection simulator (Simulation Task #3 in this figure) continuously gets the locations of all vehicles and calculates which vehicle hits against the other vehicle. if it finds a collision, a signal to inform the collision send to two vehicle simulators via Inter Simulation Communicator mechanism. Inter Simulation Communicator asks Simulation Task Manager where the two vehicles are located when Inter Simulation Communicator must send a signal. After receiving the signal, two vehicle execute each signal handler.

Two corresponded vehicle simulators should negotiate how to deal the collision signal in advance. There are several ways for handling the signal. The most simple action is to stop the vehicle itself. In this case, the vehicle simulator makes a handler to register its speed "zero" to the information storage. The other action is to trigger some other actions like calculating "crash worthiness" or "rebound" against the crash. It depends on what the users want to simulate when they choose the action against the collision detection. Whether or not which action is choose for an action of the collision signal, the corresponded two vehicle must do the same action, otherwise, an unnatural phenomenon will occur (ex. There are two corresponded vehicles in a crash. One is just stopped itself, but the other gets serious damage.) The Inter Simulation Communicator also supports this kind of negotiation.

Algorithm 3 is a sequence in a collision detection simulator and Algorithm 4 is a handler related code in a vehicle simulator. library codes to access Inter Simulation Communicator are offered by my simulation environment(line 1and 3 in this figure), therefore, all that

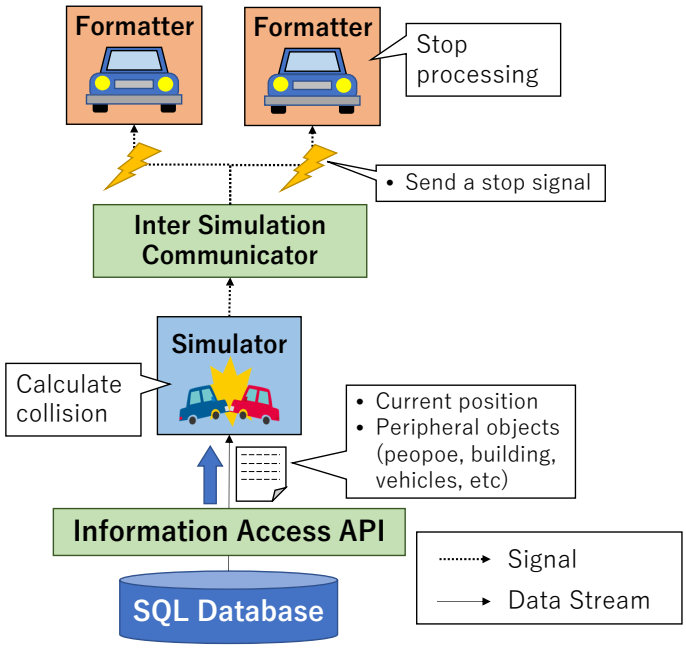

Figure 3.8: Collision detection mechanism with inter simulation communicator

user must make is the code for finding collision (from line 2 to 11). A vehicle simulator should register a signal handler to receive a collision signal before starting a simulation. During the simulation, A Collision Detection simulator continuously executes detection process for each couple of vehicles(line 5 in the Algorithm 3.

Computational complexity of the collision detection depends on how many vehicles a simulation deals with. In my simulation environment, each simulator is made as an independent executable. Therefore, the developer of a simulator can add any libraries on it (ex. a library for detecting line segment intersection in computational geometry algorithm is effect to reduce the computational complexity for finding collision detection.). On the other way, parallelizing the collision detection simulator is effect for simulator's scalability. There are many sophisticated libraries for making the simulator parallel. The

Simulation Task Manager executes the simulator on a virtual machine or container process. Some functionalities to access the other simulators or the information storage is offered as some libraries. The Simulation Task Manager set up a virtual machine environment on which a simulator is executed.

Algorithm 3 Collision Detection

Require: $id:$ *identification of this output stream*

```
1: isc ⇐ InterSimulatorCommunication.connect(id)
2: while not end of the simulation do
3:     vehicles ⇐ GetAllVehiclesInfo()
4:     for each vehicle1 ∈ vehicles do
5:         vehicle2 ⇐ CollisionDetect(vehicle1, vehicles)
6:         if vehicle2 is not null then
7:             isc.message("hit", vehicle1)
8:             isc.message("hit", vehicle2)
9:         end if
10:    end for
   (and sleep during a certain interval)
11: end while
```

3.5 Implementation

I describe a minimum set of implementation of my simulation environment in this section. This implementation focuses on seeing feasibility of my environment. I introduced the three important concept. One is the concept of an information storage to completely separate information from simulator, a simple architecture to keep connectivity between simulators, and multiple participation of users and AIs. Considering the three concepts, "delay between the components" and "how many participants can join" should be the evaluation points to prove the feasibility of my simulation environment. Separation of

Algorithm 4 Collision Message Handler

Require: id : $identification\ of\ this\ output\ stream$

```
1: isc ⇐ InterSimulatorCommunication.connect(id)
2: isc.RegisterHandler(id, Handler)

   (message handler)
3: procedure HANDLER(msg)
4:     if msg == "hit" then
5:         (stop this vehicle)
6:     end if
7: end procedure
```

information from simulation environment generates time delay to access the information. It finally causes time gap between simulators and I referred the fact how influences the timing gap in the "Simplify connectivity and concurrency support" section. Therefore, I must see the point whether or not the time delay to access the information is enough small not to influence a simulation.

Figure 3.9 shows the implementation overview. It consists on three parts. One is the functions in an user PC, another is the functions in a simulator PC, and the other is the Information Storage PC.

Inside the the user PC, a wheel and a pedal are connected via USB cable, and ROS [45] receives the manipulated variables from the USB devices. ROS is a set of software development environment and libraries for robot. ROS send the manipulated values to the vehicle simulator with GStreamer [1]. GStreamer is a library set to implement streaming software. I use the ROS and GStreamer at this moment, instead of forwarding the manipulated variables directory from the USB devices to the pseudo devices because it simplifies

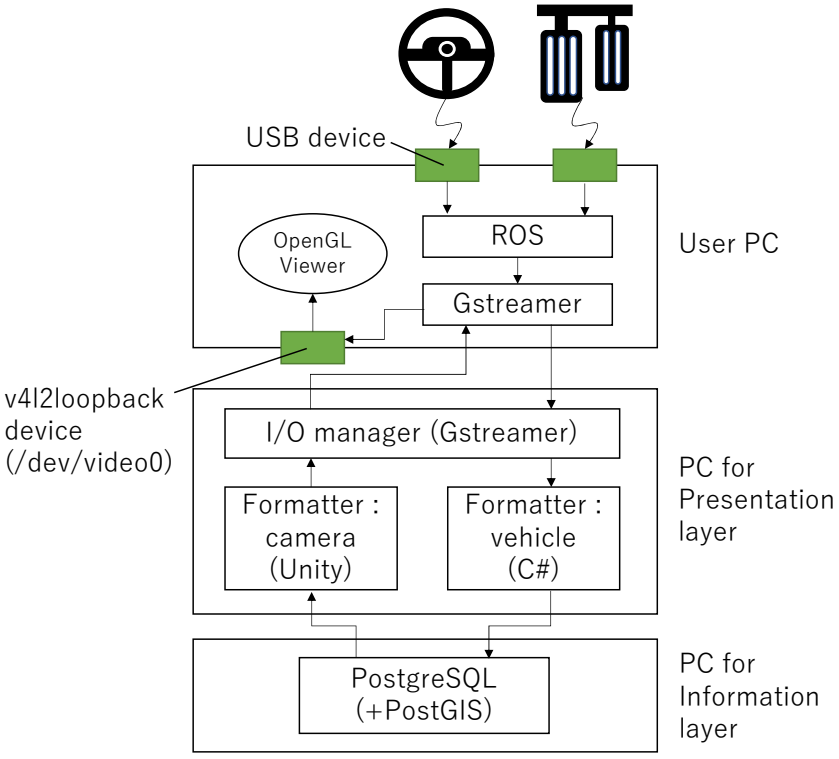

Figure 3.9: Block diagram of the basic implementation

the implementation, but ideally, row data from the USB devices should be forwarded to the vehicle simulator and translate the row data into the format which the vehicle simulator store to the information storage.

Inside the Information Storage PC, I use PostgreSQL server as the information storage and add PostGIS package to deal with geometry information. Figure 3.10 shows the map related figures. Left side of the figure describes the geometry information of buildings and roads. I import map information of a certain area from Open Street Map [4]. PostgreSQL has a program to translate the Open Street Map format into PostGIS format (osm2pgsql). The geometry information created from Open Street Map consists several numerical values, strings, linear data and polygon data. The linear data represents the shape of roads and polygon data represents the shape of buildings or some other structures (ex. parking etc). Above camera simulator issues a query with range specification with ST_Dwithin option to get these information.

Inside the simulator PC, there are two simulators. One is a camera simulator. Another is a vehicle simulator. I made the camera simulator with Unity and the vehicle simulator with Java. The camera simulator get the peripheral information from the storage first. The peripheral information with range option has several kinds of structures. I only choose buildings and roads for a starter simulator. As a simple technique for visualization of buildings, it is feasible to use "asset" of buildings. Asset means the data set which represents some structures, especially indicates 3D object data. Otherwise, we must define the each buildings individually and it would not be feasible. When a camera simulator visualize the peripheral information, the most important thing is to create a "plausible view", not to make the strict view of real world. The detail of the buildings (ex. the strict

layout of the windows, etc) would not affect to make an AI wise. Therefore, to use asset of building on Unity is an appropriate way to make a plausible view quickly I think.

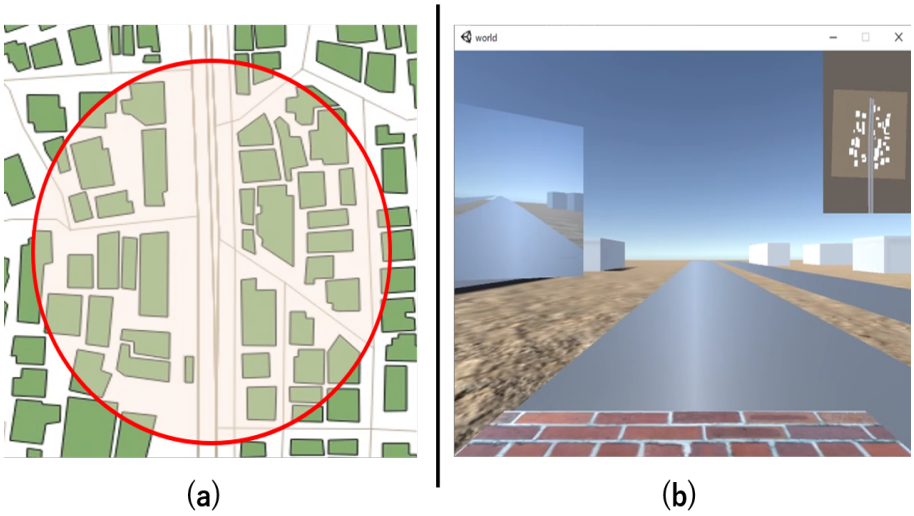

Figure 3.10: Camera view (a) map information from the information storage with GIS format. The red circle represents the range that the camera simulator get as the peripheral information. (b) Rendering result of the geometry information by Unity

Most of the building assets are rectangle shaped. To instantiate 3D buildings with an asset in Unity, the developer must input the parameters based on rectangle shape (width, height, depth, position, etc). On the other hands, as you see the figure 3.10 (a), there are many non rectangle shaped buildings in the map or slightly distorted rectangles, so I create a largest inscribed rectangle in a polygon with the method [9] and input the parameter of this inscribed rectangle to instantiate 3D buildings. There are several white cubes in the figure 3.10(b). They represent buildings which are instantiated with the parameters of the largest inscribed rectangle, so it is easy to replace the white cubes with some building assets. Open Street Map data has no height information of buildings. I define the height

data with random value for the time being.

3.6 Preliminary Evaluation

In this section, I am going to give preliminary evaluation to my implementation. The objective of this research is to separate a monolithic simulation environment into many parts and introduce well-known technology like virtual machine or SQL database. Such distributed simulation environment acquires interoperability and scalability. On the other hand, my simulation environment makes additional communication time between the simulation components, compared with a monolithic simulation. Referring to the figure 3.9, There are three computers, and they execute the individual simulation components. When a camera simulator acquires data from PostgreSQL, network access delay will occurs between them. It is the same situation between a vehicle simulator and PostgreSQL. a network communication is also seen between I/O manager and GStreamer on a User PC. Therefore, the evaluation should be focused on the network communication time between the simulation components.

3.6.1 Three Evaluation Cases and Environment

I evaluate the three cases. One is the communication time between two simulation components which are executed on a same physical machine. Another is the communication time in the case that each simulation component is executed on a different virtual machine. The other is the communication time in the case of different physical machines. I also choose two components from three places depicted on the figure 3.9 ; from PostgreSQL to Camera

formatter, from Camera formatter to GStreamer, and from ROS to PostgreSQL. Therefore, the evaluation has nine combination of multiplying 3 patterns of computer usages by 3 patterns of measurement points. The table 3.2 is the summary of evaluation points. I also describe the evaluation environment in the table 3.1.

Table 3.1: evalustion environment

physical machine	CPU	Core i7-7700 3.6GHz 4 core
	memory	16.0 GB
	disk	SSD 500 GB
virtual machine	VMM	Oracle VirtualBox
	CPU	Core i7-7700 3.6 GHz 2 core
	memory	4.0 GB
	disk	virtual disk (vdi) 100 GB
network	wired 1Gbps ether	

Table 3.2: Evaluation summaries

Case 1	use one physical machine		
	measurement point	from	to
		PostgreSQL	Camera formatter
		Camera formatter	Gstreamer
		ROS	PostgreSQL (via Vehicle simulator)
Case 2	use two virtual machines on one physical machine		
	measurement point	from	to
		PostgreSQL	Camera formatter
		Camera formatter	Gstreamer
		ROS	PostgreSQL (via Vehicle simulator)
Case 3	use two physical machines		
	measurement point	from	to
		PostgreSQL	Camera formatter
		Camera formatter	Gstreamer
		ROS	PostgreSQL (via Vehicle simulator)

This evaluation reveals the two concerns. One is whether or not the communication time I measured is affordable for simulation infrastructure of self-driving system. Another

is the appropriate place on which a simulation component should be run to minimize the communication time. I will discuss the above two concern in the section 3.7.

3.6.2 Result

The evaluation result is described in the figure 3.11. The x axis indicates three places where the communication time is measured. The y axis indicate the communication time with millisecond order.

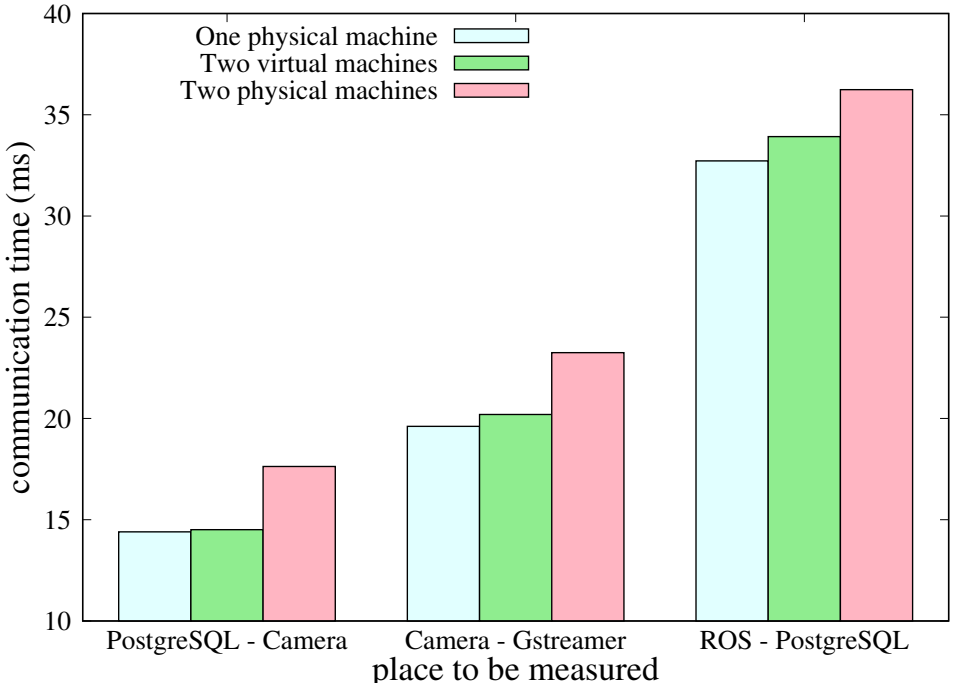

Figure 3.11: Evaluation

First of all, I explain differences of the communication times measured at three places. One is the place between PostgreSQL and camera formatter. the communication time is much smaller than the other two places. the camera formatter visualizes the peripheral

place on a map with the object data acquired from the database, so the communication includes the object data. In the case of this measurement, it includes 30 objects data and one object is composed of several bytes of the numerical values. The communication time between the camera formatter and GStreamer is longer than the previous place because the communication includes the camera view data created by the camera formatter. the camera view image is much larger than the object data but the communication time gap is about 5ms because the data is transferred within the memory in a same physical machine. Compared with the previous two place, the communication time between ROS and PostgreSQL is the most large because the communication data is transfer via several simulation components.

Next, I explain the gap caused by the measurement at the three circumstances that two simulation components are executed on one physical machine, two virtual machines or two physical machines. In the case of two physical machines, the communication time is longer than the other two circumstances because the communication between two physical machines uses the physical network facility, in contrast with the other two communications just use several main memory for data transfer. Several network stacks are skipped in the case of the two circumstances (one physical machine, two virtual machines executed on one physical machine) because they do not use an actual network route, so they do not need to make an actual network packet. About the comparison of the two circumstances (one physical machine, two virtual machines executed on one physical machine), there are no huge gap between them. The number of functions that a communication data pass through in the case of two virtual machines is larger than the case of one physical machine, so the processing time in the case of two virtual machines should be longer than the case of one physical machine. However, the two circumstances have almost same communication time.

A modern Linux kernel processes a network packet without memory copy and the most part of the network processing time in a Linux kernel is to copy memory data, therefore, to reduce the number of memory copy also reduces the most part of the network processing time. I estimate the reason why there are no huge gap between two circumstances is the result of the network packet processing in Linux kernel without memory copy.

3.7 Discussion

Referring to the result described in the section 3.6.2, I discuss the two points. One is whether or not the communication time I measured is affordable for simulation infrastracture of self-dring system. Another is the appropriate place on which a simulation component should be run to minimize the communication time.

There are no obvious requirement how much communication time (in other word, it would be called "response time") is affordable for self-driving system, but there are some researches that deal with the impact of delay on multiplayer games. The literature [41] is one of such researches. According to the literature, the delay can hardly be noticed up to 50 millisecond and acceptable up to 100 millisecond, if no high demands with respect to realism are needed. Looking at the result described in the figure 3.11, the communication time measured at all places are not exceed 40 millisecond. Therefore, my simulation infrastructure can execute a self-driving simulation without any effect caused by the communications. However, this preliminary evaluation is "ideal" for measurement of communication time because there is no other components that prevent the simulation components from smooth network communication and a lack of CPU time does not occur. In an actual simulation, there are huge amount of simulation components that randomly make network connections

and that will increase the network communication time. **Consequently, I realized the fact that it is the most important how to deploy the simulation components in a cloud.** An appropriate deployment of the simulation components implements smooth network communication and that prevents my simulation infrastructure from degrading the simulation capability.

3.8 Conclusion of this chapter

In this chapter, I suggested a distributed simulation environment that simulator and formatter is connected via a middleware and all data is stored in one database and shared to all simulation components. my simulation environment enable us to develop a simulator independently and swap it to another one in accordance with a test case. I also gave a preliminary evaluation to my simulation environment and it reveals the fact that my simulation environment is feasible for self-driving simulation although it has additional communication time between simulation components. I would like to suggest a novel deploy mechanism in the next chapter.

Chapter 4

A Deploy Mechanism using Voronoi Dividing for Scalable Geographical Simulation

In this chapter, I suggest a novel virtual machine deployment mechanism for virtual machine based simulation environment. A vehicle sometimes has a network connection with another vehicle which runs nearby to exchange the positional information. Eventually such network connection makes large scale ad hoc network. However, unknown network topology shaped by the ad hoc network create a partial network congestion. Therefore, it should be better to deploy a group of virtual machines in one physical machine, which are "geographically" close to each other. I explain my deployment mechanism in the following sections.

4.1 Problem

The existing VM deployment mechanism is not appropriate for the large-scale VM-based ad hoc network simulation. In Figure 4.1, I depict a typical case wherein partial network congestion occurs. From the figure, we can see five simulated vehicles (A–E) located on a map, in accordance with a simulation scenario. Each vehicle establishes connection with another sufficiently close vehicle to send a packet. Consequently, an ad hoc network is finally created with its topology described using short dash lines in this figure. However, VMs (described using rectangles A–E), which represent simulated vehicles, are deployed using Round Robin algorithm on PM1 to PM3.

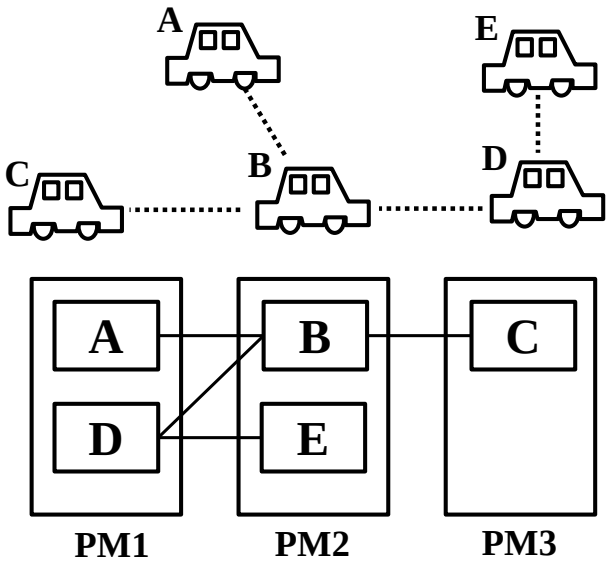

Figure 4.1: Local concentration of network sessions in the case of Round Robin based VM deployment

In Figure 4.1, I depict the case wherein there are considerably more network connections between PM1 and PM2 than those between PM2 and PM3. The existing deployment

mechanism does not consider the network topology that is created by the ad hoc network protocol, as it cannot estimate the network topology in advance; i.e., it cannot estimate which vehicle will establish a network connection with which vehicle. Ideally, VMs B, D, and E should be executed on the same machine to reduce the number of network connections over the physical machines. However, there exists no appropriate deployment algorithm to minimize the number of network connections between physical machines. The increase in the number of network connections between physical machines is approximately equivalent to the network-bandwidth saturation of the physical network. The network-bandwidth saturation limits the scalability of the vehicular ad hoc network simulation; therefore, I would like to put as many as possible network connections into a physical machine.

4.2 Geographical Distance based Virtual Machine Deployment

As I mentioned the fact that existing deployment approach like round robin occurs network congestion because the existing approach deploys the virtual machines without knowledge the network topology. In another word, none of existing deployment mechanism estimates the network topology, before an ad hoc network simulation is executed.

Figure 4.2 describes the basic concept of my deployment approach. Left-side of this figure describes a traffic simulation and right-side describes a computing cloud. In my deployment mechanism, the simulated map is separated into several parts and one physical machine is assigned to each part of the map. A virtual machine represents simulated vehicle is executed on a physical machine which the same part of the map is assigned. In the figure 4.2, Area #1 is assigned onto PM1 and two virtual machines are executed on PM1. They

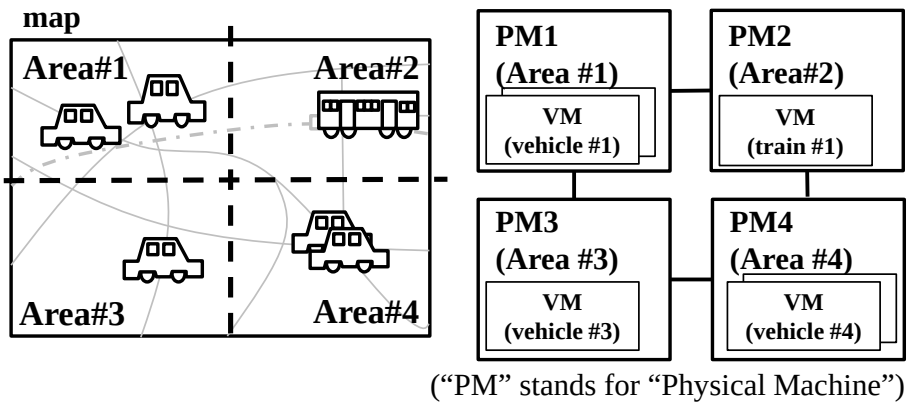

Figure 4.2: dividing the simulation map into several areas and assign them into each physical machine

are appeared on the Area#1. If a simulated vehicle crosses over the boundary of the area. The virtual machine represents its simulated vehicle is also moved to another physical machine (typically VM migration is used).

4.3 Voronoi Decomposition of Simulated Map

How to separate the simulation map into each area is one of the key factors for reducing the network sessions between physical machines. Vehicles basically stay at a junction longer than the other place. It could be easily imagine that vehicles at a junction have much more network connections, compared with the other place. If we separated a map on a junction, VMs would have many connections between the physical machines and it leads to network congestion. Therefore, junction should be kept away from the boundary of an area as far as possible.

I introduce Voronoi decomposition as a separation method of a map. Voronoi decomposition is a partitioning method of a plane based on the distance of each point. Voronoi decomposition makes perpendicular bisectors between adjacent two points on a plane and

it becomes the boundaries of an area and finally makes tessellation of a plane. Voronoi decomposition has an unique characteristic that all of the points equally have maximum distance from the boundary. I regarded a junction of a simulation map as a point of Voronoi decomposition.

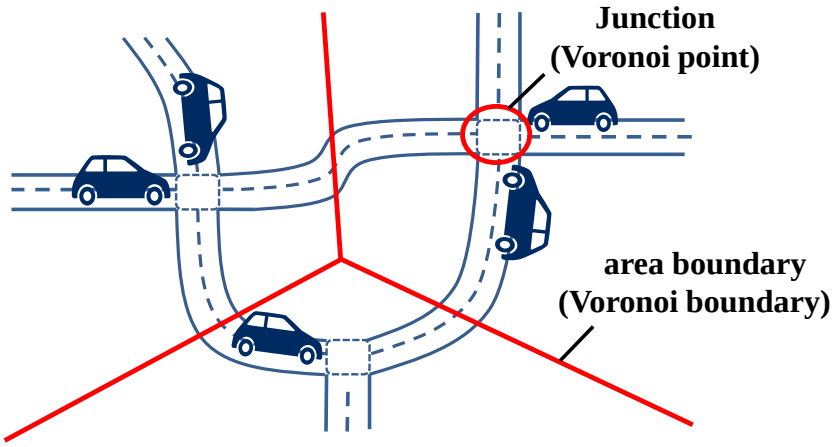

Figure 4.3: voronoi dividing based on junction

Figure 4.3 is an example of Voronoi decomposition of a simulation map. There are three junctions as Voronoi points and make perpendicular bisectors between them. Each junction equally has maximum distance from the area boundary. Considering the fact that vehicles stay at a junction longer and make much more network connection around a junction, Voronoi decomposition method is effectively put most of network connection into a physical machine. Practically, we should assign a physical machine to a certain group of areas because we can not assign a physical machine to each junction. I call a group of the areas as **"area cluster"** in the following sections.

4.4 Load Balancing

A physical machine is assigned to a certain area cluster on which some simulated vehicles are running. When a lot of simulated vehicles get together on a certain area cluster, I must add another physical machine for load balancing. On the other hands, all of the simulated vehicle has gone from a certain area cluster, the physical machine which is assigned to its area cluster should be stopped working (and also some other physical machines must take over its area cluster). In this subsection, I explain a basic load balancing method of my approach. I used the number of VMs on a physical machine as a threshold to execute the following method.

4.4.1 Divide an Area Cluster, Add a Physical Machine

When a lot of simulated vehicles get together on a certain area cluster, I must add another physical machine for load balancing on this area cluster. These four sequences are the basic procedure to separate an area cluster and assign physical machines on two new area cluster.

i. divide the set of vehicles on the target area cluster into two groups. It is based on the position of each vehicle. I chose X axis (longitude in the simulation) for listing, sorting and grouping of the simulated vehicles.

ii. also divide the area cluster into two clusters based on the two set of vehicle groups.

iii. If there are two vehicles on one area and each vehicle belongs to a different group (It is possible because two new area clusters are made, based on the number of simulated vehicles). This area belongs to the area cluster which has less simulated vehicles than

another area cluster. If the two area cluster has equal amount of vehicles, this area belongs to one of them randomly.

iv. migrate the VMs to a designated physical machine. A physical machine which is assigned to the old area cluster will be assigned again to an new area cluster. Some VMs keep staying on. and some other VMs move to a new physical machine which is assigned to another new area cluster.

If many simulated vehicles are stacked on one certain junction, a physical machine will be added on that area without consideration of increasing of network connections, so that I deal with this situation with normal deployment manner. As I said it before, we can not estimate the ad hoc network topology beforehand, therefore, there are no appropriate criteria to separate one primitive area into multiple areas.

4.4.2 Breakup an Area Cluster, Remove its Physical Machine

If all of the simulated vehicle has gone from a certain area cluster, we don't need the physical machine which is assigned to its area cluster. On the other hand, the areas in this area cluster should be taken over by the other area clusters for preparation of the future when a vehicle goes into one of these areas again.

I use Delaunay diagram to recognize the neighbor area. Delaunay is a diagram that each point on a plane makes a line to the neighbor point. Delaunay diagram indicates the positional relationship which area is an neighbor of the other areas in Voronoi diagram. The following is the basic sequence to breakup an area cluster. I call an area cluster "AC_breakup" which is about to be broken up and "AC_peripheral" which is a neighbor area cluster of "AC_breakup".

i. search peripheral areas of AC_breakup and recognize AC_peripheral the peripheral areas belong to.

ii. measure the distance between the peripheral areas and each area in AC_breakup. As a result of this measurement, each area in AC_breakup finds a closest peripheral area in and AC_peripheral.

iii. finally, each area in AC_breakup is merged into a AC_peripheral which the closest peripheral area belongs to.

To breakup an area cluster with Delaunay diagram sometimes make a sort of "enclave". An enclave increases the number of VM migration between physical machines. I don't care these enclaves in the breakup sequences, but I offer a method to "reorganize" the area clusters later in this section.

4.4.3 Partial Exchange of Area Management

Dividing an area cluster is a heavy task because a lot of VM migration occurs. To reduce the dividing task as many as possible, I introduce partial exchange of area management. When a vehicle goes over the border of an area cluster, this method chooses the alternative plans.One is to execute VM migration. Another is to exchange their own areas partially. This method works as a constant load balancing method. In contrast with this method, dividing works as an urgent load balancing method.

Figure 4.4 describes a typical case that partial exchange is occurred. Vehicle#1 is about to go over the boundary of two area clusters named "AC1" and "AC2" and Vehicle#2 is running on AC2. In this case, if Vehicle #1 moved to AC2, AC2 would have two vehicles and AC1 would have no vehicle. Therefore, partial exchange of the area should be done

in this case. The following sequences are the procedure to partially exchange their own areas.

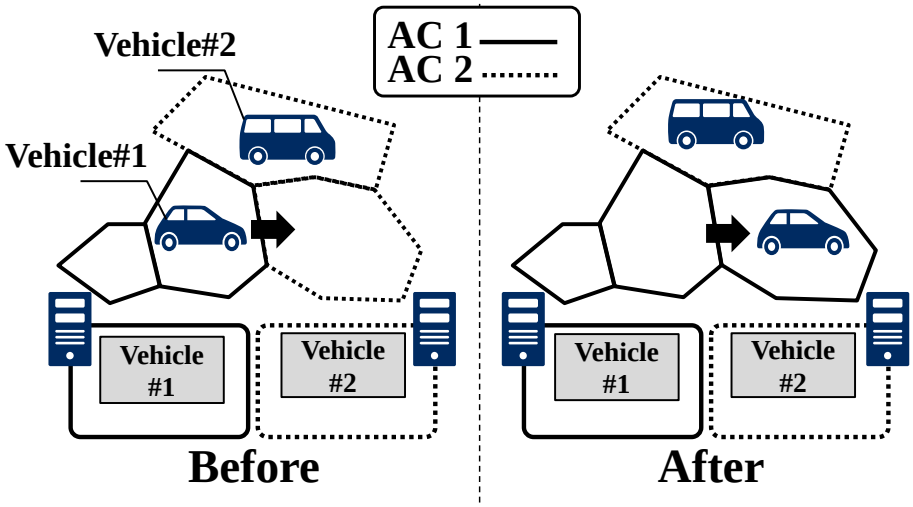

Figure 4.4: partial exchange of area management

i. When a vehicle is about to go over the boundary of two area clusters, my mechanism counts the number of vehicles on each area cluster first. Meanwhile, my mechanism always monitors the vehicle's position and watches a border crossing for making a decision whether or not partial exchange of areas should be done.

ii. Compare the next two states. One is the state after partial exchange is done. Another is the state after two area clusters keep their own areas and the vehicle is migrated. In the case of figure 4.4, if the partial exchange is done, AC1 will have one vehicle (Vehicle #1). AC2 will have one vehicle (Vehicle #2). In contrast, if they keep their own areas, AC2 has two vehicles and AC1 does not have any vehicles.

iii. One of the above two states is chosen. In the case of figure 4.4, partial exchange the target area (Vehicle #1 is about to come) is executed. Finally, the target area

belong to AC1.

I consider that an option to share an area with two area clusters is not an appropriate way for the time being. To share an area equals to increasing network connections between two physical machines. In the case of figure 4.4, if AC1 and AC2 shared the area, Vehicle #1 would not move. Instead, Vehicle #1 and #3 would made a network connection between the two physical machines. Not to share also increases the number of movement of VMs. There is no deterministic way to choose which is better to reduce the network bandwidth, share or not share, therefore, I choose not to share an area in the aspect of the reducing the network connections.

4.4.4 Reorganize the Area Management

As the result of the above operations (dividing, breakup an area cluster and partial exchange of area management), my approach sometimes makes "enclaves". Enclaves mean that an area which belongs to a certain area cluster is surrounded by the other areas which belong to the other area clusters. If a vehicle passes over an enclave, many VM migrations must be executed and these migrations take a lot of CPU time. These enclaves gradually degrade the cloud performance, so they should be removed. Removing enclaves has two effects. One is to reduce the network connections between the physical machines. Another is to reduce the number of VM migrations. Reorganization of the area management should be done as a regular task with constant time interval. This task is similar to disk defragmentation. The following sequence is the basic sequence to remove some enclaves.

i. Recognize enclaves of an area cluster at first. If an area cluster is separated to several area clusters, some of them are "enclaves". The biggest area cluster should be the

main. If Delaunay diagram can not connect all of the areas which belong to one area cluster, there should be some enclaves.

ii. For each area cluster, make a convex polygon of junctions which consists of the outermost areas of each area cluster. This procedure should be done in the ascending order with the number of VMs.

iii. If a polygon includes some areas which belong to one of the other area clusters, these areas are merged to the area cluster.

iv. VMs in the merged areas are moved to the physical machine of the new area cluster.

v. Repeat the sequence 2 and 3 until there is no enclaves

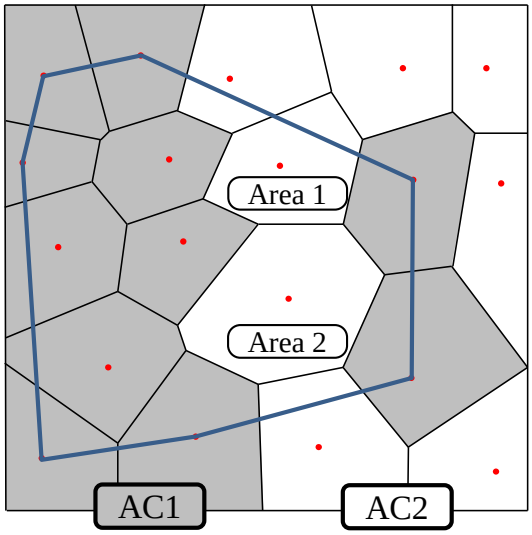

Figure 4.5: enclave

Figure 4.5 describes typical case of removing some enclaves. There are two area clusters named "AC1" and "AC2". Gray areas belong to AC1 and White areas belong to AC2.

AC1 has two enclaves and AC2 has no enclave. Assuming that AC1 has less vehicles than AC2, AC1 makes a convex polygon earlier than AC2 and the convex polygon is figured on the figure 4.5. In this case, "Area1" and "Area2" which belong to AC2 is merged to AC1. In contrast with the above case, if AC2 has less vehicles than AC1, AC2 makes a convex polygon earlier than AC1. The two enclaves which is located at right-side of Area1 and Area2 are merged to AC2.

4.5 Simulation Architecture

Figure 4.6 describes the architecture of my simulation environment. The environment consists of a Simulation Manager, Traffic Simulator, Cloud Manager, WiFi Emulator and Some VMs on the physical machines (Node 1 to N on the figure). Traffic simulator is executed at first and it produces each vehicle's position on every clock, then the map and each vehicle's positions are put into both simulation manager and wifi emulator. Wifi emulator calculates the packet drop rate of each vehicle on every clock and set it on the network interface card on each VM. Simultaneously, Simulation Manager makes Voronoi diagram from the simulated map at first and then configures a number of area clusters. The number of area clusters depends on the number of physical machines to be used during initialization. Then these physical machines are assigned on each area cluster. Finally Simulation Manager requests VM deployment/migration to a Cloud Manager in accordance with each vehicle's position on every clock. When the Cloud Manager deploys a VM, it informs IP address of each VM to Wifi emulator to recognize which VM represents which vehicle on the traffic simulation.

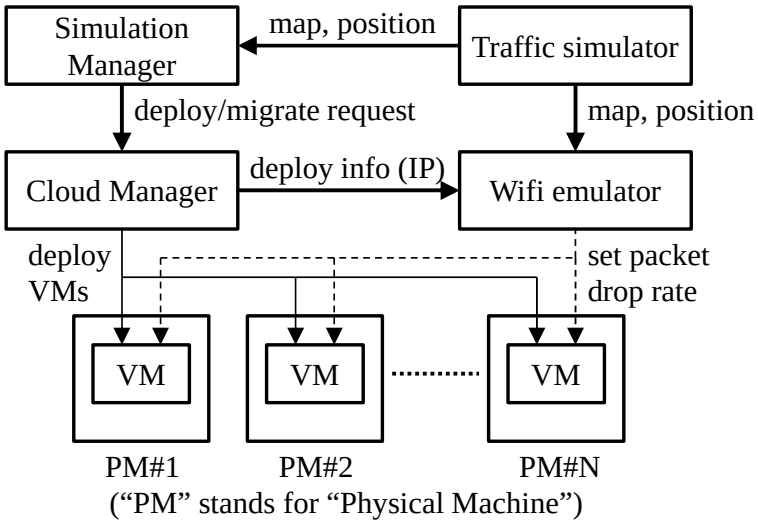

Figure 4.6: simulation architecture

4.6 Evaluation

I suggested a VM deployment mechanism which reduces network connections between physical machines on an ad hoc network simulation based on virtual machine. On the other hands, my approach uses physical machines much more, compared with the existing deployment mechanism because my approach do not always pack VMs into a physical machine as much as possible. In another word, the number of running vehicles in an area cluster do not always equal to upper limit of VMs running in the physical machine. Therefore, the evaluations of my mechanism should have two points. **1. how many network connections between physical machines on a cloud can my approach reduce? 2. how many physical machines will my approach be used, compared with existing deploy mechanism?** I would like to explain the evaluation environment first, and then show the result of the above two evaluation points in this section.

4.6.1 Evaluation Circumstances

Table 4.1 is a list of programs which I used in this evaluation. I made Simulation Manager and mock-up modules which represents WiFi emulator, cloud manager and VM and physical machines. I also used SUMO traffic simulator [34] to create the input data (a map and positional information of each vehicle). The programs in the Table 4.1 are made with Python, except for SUMO traffic simulator. Simulation Manager requests deployment or migration of a VM to Cloud Manager. Cloud Manager creates a class instance represents a VM, instead of creating an actual virtual machine. An ad hoc network is basically created between two vehicles which are enough close to each other, so that WiFi emulation mock-up module directory calculates network connection information from the positional information of each vehicle on every clock, which vehicle has a network connection with which vehicle, instead of calculating packet drop rate.

Table 4.1: Programs to be used for evaluation

Simulation Manager	My program
Traffic Simulator	SUMO traffic simulator
Wifi Emulator	Mock-up module on my program
Cloud Manager	Mock-up module on my program
VM, physical machine	Mock-up module on my program

One of the most important things is to reveal the fact that "ideally" how many network connections between physical machines can be reduced by my approach. Therefore, I think that the evaluation should be isolated from actual operation of a cloud management. To evaluate my approach without dependency of a specific cloud management, I do not use an actual cloud environment like OpenStack, Kubernetes and so on, but my mock-up module for VM and physical machines can be replaced easily to an actual one.

Table 4.2 describes the parameters used by SUMO traffic simulator. SUMO traffic

simulator needs a map and vehicle's information (appealing place and route) as input. I made a grid-shaped load map and vehicle's information with the scripts which is a part of SUMO simulator. maximum 1000 vehicles are generated at the beginning of the simulation and they run with random walk until the end of the simulation.

Table 4.2: Traffic simulation parameters

map	$2km \times 2km$ square, 400 junctions, two-way loads
number of vehicles	100 to 1000 vehicles
simulation time	200 seconds
each vehicles emersion	beginning of the simulation
route of each vehicles	Random

4.6.2 Evaluation

Physical machine utilization

Figure 4.7 indicates the result of evaluation on the utilization of physical machine. x axis means the elapse of simulation time and y axis means the number of physical machines used in this simulation. This evaluation is the comparison between complete packing and my approach. Complete packing is a deploy approach which completely pack VMs into each physical machine until it reaches the upper limitation of the number of VMs on one physical machine. Therefore, it means an ideal situation in the point of CPU resource utilization. I set the upper limitation to 10 VMs. If 10 VMs are executed on one physical machine, another physical machine will be used for the next VM. This result obviously indicates the fact that my approach uses physical machine much more than complete packing during the simulation. Especially, my approach uses them at most 2.5 times than complete packing approach.

There are two reasons of this gap. One is that my approach does not always use a

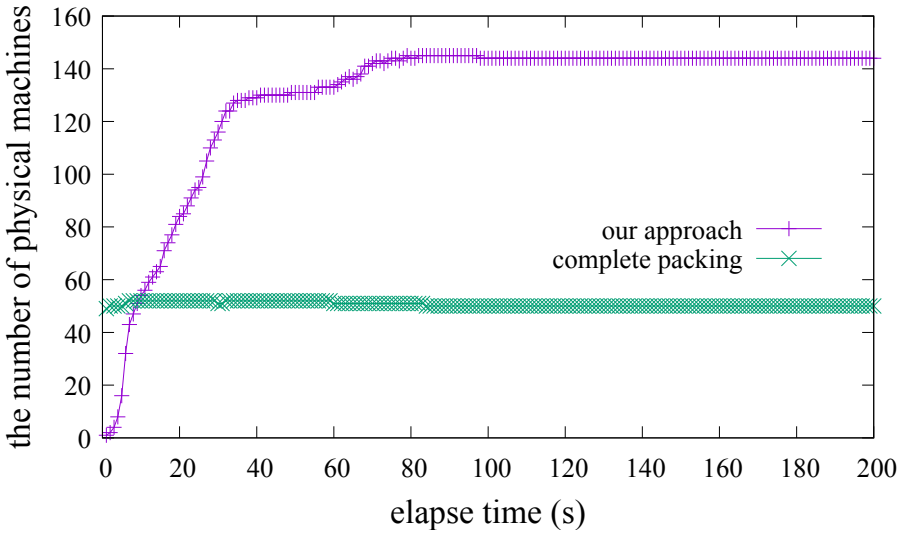

Figure 4.7: The number of physical machines to be used to execute 500 VMs

physical machine completely until its limitation. Another is that my approach does not breakup an area cluster and remove the physical machine until when vehicles running on that area cluster becomes zero. To wait it until zero is the simplest way because none of vehicles has to move to another physical machine, meanwhile if my approach had a threshold to breakup the area cluster, a certain number of VM migrations would be executed and it would take additional CPU time. There is a trade-off between the number of VM migration and the utilization of a physical machine. I do not have an appropriate answer when my approach should breakup an area cluster for the time being.

Number of sessions between physical machines

Figure 4.8 indicates the relationship of elapse of simulation time and the number of network sessions between the physical machines. This result is a comparison between round robin, complete packing and my approach in the case of 500 VMs. Looking at the figure 4.8, Characteristics of round robin and complete packing is almost same during the whole

simulation time. They have about 300 network sessions constantly during the simulation. On the other hands, in my approach, the number of network sessions reached about 50 in this simulation, roughly my approach reduces the network session over the physical machine by one-sixth.

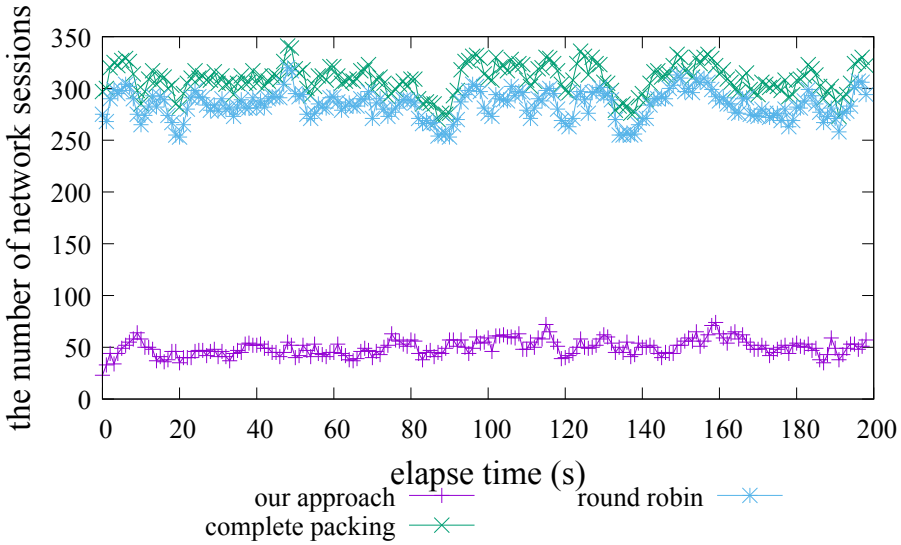

Figure 4.8: Elapse of the number of network sessions on every seconds (executed 500 VMs)

Figure 4.9 indicates the relationship between the number of VMs and network sessions. x axis means the number of VMs and y axis means the total number of sessions over the physical machines. Around 500 VMs in this figure, the gap between the existing approaches (round robin and complete packing) and my approach is about six times, and the difference between my approach and the existing approach is getting bigger as the number of VMs increased. In contrast, my approach controls the increase of network sessions with almost linear shape. Therefore, these result show us the fact that my approach can drastically reduce the network sessions, compared with the other approaches.

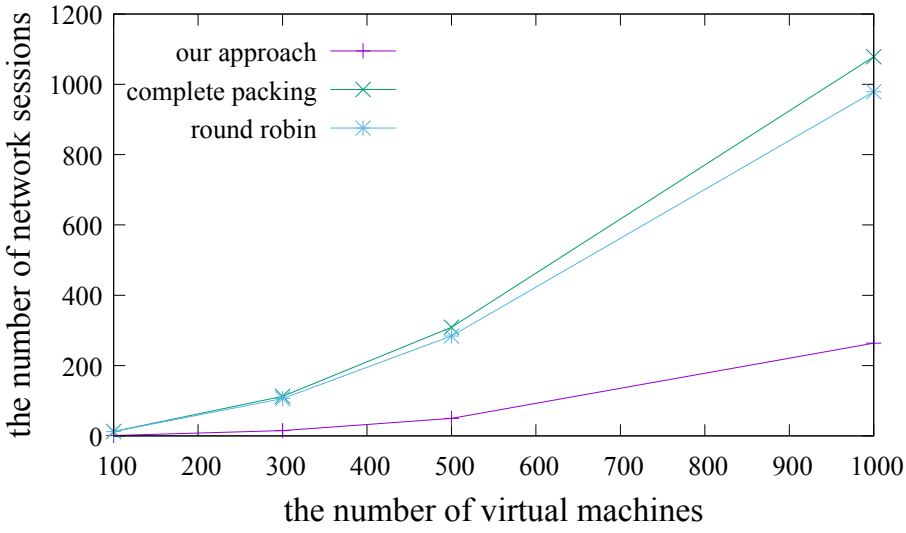

Figure 4.9: The total number of network sessions

4.7 Discussion

Is there any approaches to increase the CPU utilization? : As I mentioned this topic in the section 4.6.2, it is a key factor how to make a threshold to breakup an area cluster. My approach basically wait to breakup until there is no VM on the area cluster, but it makes low CPU utilization. On the other hand, if I set a threshold on a certain number of VMs, My approach must execute a certain number of VM migration and it might gain network latency if some of the VM during migration has network session or about to start network session. This kind of network latency should be isolated from simulation because this is a problem of simulation environment. I am searching for an appropriate approach to find an optimal threshold.

Can my approach reduce more network session over the physical machines? : Dividing an area cluster and add a physical machine (section 4.4.1) makes a certain amount of VM migration. If two VMs (one is about to migrate to a new physical machine

and another keep staying at the current machine) have an network connection, this division creates more network sessions over the physical machines. Meanwhile, there is a possibility that these VMs might disconnect the current network session and make a new network session with the other VMs on the same physical machine after the division. In this case, the gain of network session during division is transient. Eventually, my approach should predict where the vehicles go, if my approach must reduce more network session during division, but effectiveness is ambiguous.

Therefore, the "dividing" and "breakup" methods of my approach still need to be researched to find an optimal threshold.

4.8 Conclusion of this chapter

I suggested a deploy mechanism of VMs which was based of Voronoi decomposition of a simulated map. My mechanism executes several virtual machines on a same physical machine, which are " geographically " close to each other. My approach reduces the number of network sessions by one-sixth in the case of 500 VMs, compared with complete packing and round robin dispatching. We're convinced that my mechanism contributes to embody a scalable ad hoc network simulation.

Chapter 5

A Priority Control Method based on Predicting The Simulation Events

5.1 Problem

Simulation clock synchronization on all of the processing objects is most important for this loosely-coupled simulation environment. Simulation clock means a clock system which is embedded in a simulation program. My simulation environment is categorized as an agent-based model simulation that a bunch of independent agents do actions and have interactions each other. As a result of the simulation, I can observe and get a new knowledge from the behavior of agents. Traffic (traffic jam) simulation, Social network simulation and pedestrian simulation are the examples of agent-based model simulation. In an agent-based model simulation, every agent must keep simulation clock to be synchronized, otherwise, the simulation will lose credibility. In the case of traffic simulation, car crash which must be happened does not occur because of clock skew.

There are several techniques to synchronize the clock in an agent-based model simulation. One of the most obvious technique is to have a "global virtual time" [22]. Global Virtual Time (GVT) is a one of a clock system to be shared between all of the processing objects appeared in a simulation. Once a GVT is set, all of the processes should be obey the GVT. GVT and the other synchronization techniques are often discussed and used for parallel simulation. However, these synchronization technique would be too heavy to implement on loosely-coupled simulation environment which is based on virtual machine technique because clock synchronization cost over the VMs is much higher than a parallel simulation. GVT also degrades the scalability of simulation environment.

It is difficult to have GVT in loosely-coupled simulation environment, therefore, keeping real-time processing in a VM would be a practical solution. Almost all computer has a hardware timer. Operating System (OS) constantly makes interruption with this hardware timer and then increments a variable by one. For example, if a hardware timer is set to every 10 millisecond, timer interruption will be occurred and then a variable inside OS (it is called jiffies on Linux) will be incremented every 10 milliseconds. On the other hands, if a CPU is not assigned to a certain VM, the variable in this VM is not incremented. This means the fact that the clock inside the VM gets late and some tasks based on the timer also get late. As a result, a car crash which must be happened does not occur because of clock skew. To prevent this situation, enough CPU always should be assigned to a VM and keep time continuity inside the VM.

However, there is a case that lack of CPU is spontaneously happened during a simulation in out environment. Figure 5.1 describes a typical case that a part of simulation in a VM is getting late. Left side of this figure describes a simulation map and vehicles.

Right side describes task scheduling with timeline. I explained the deploy mechanism of my simulation environment in the figure 4.2. Single or multiple physical machines are assigned to each area on a simulation. As a simulation goes on, the number of vehicles in an area also vary. At this time moment, there are four vehicles in the left side of the figure, so four VMs should be executed on the physical machines which is assigned on this area. This physical machine has only two CPUs, so the VM execution will be delayed because of the lack of CPU. In this case, time lapse is twice as fast as a clock inside a VM (when the real clock is T=2, VM executes the task A and task B of t=1).

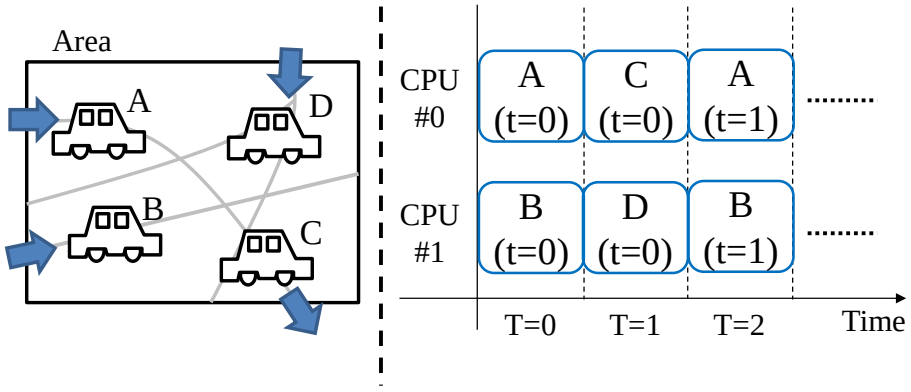

Figure 5.1: Problem

In Conclusion of this section, the biggest problem of my simulation environment is how to keep real-time processing under the spontaneous increase of VMs in an area.

5.2 Related Works

I will see two categories of related works in this section. One is task scheduling for cloud computing, Another is virtual machine based simulation environment. I investigate how to keep real-time processing on a cloud in the former category and also check how to

synchronize simulation clock between the whole VMs on a similar simulation environment.

5.2.1 Task Scheduling for Cloud Computing

There are many works related to task scheduling on cloud computing. Task scheduling on cloud computing is basically for reducing turn around time (TAT) of a request which comes from the outside of a datacenter and its technique is based on how many request assign to each computer with a certain criterion (request priority, energy aware, high CPU utilization etc). Task scheduling related works are categorized by Mathew et al in the literature [37].

Load imbalance on each computer in a datacenter is caused by two reasons. First reason comes from the imbalance of the number of requests which each computer have to process. Second reason comes from the imbalance of the processing time of each request. If it is assumed that the processing time of a request is completely equal to all computer and arrival interval of each request is equal, round robin scheduling is the only answer to keep equality between all request's TAT. But practically, processing time is not equal to all requests, and arrival interval is also not equal, so task scheduling algorithm based on mathematics is needed.

My simulation environment does not receive any requests from the outside, but causes the imbalance of processing time on each VM. Each VM executes self-driving AIs with several sensors. Each AI has different algorithm which needs different processing time, and also processing time on a certain AI vary in accordance with input data. For example, object recognition from camera view is a typical case. If there are few vehicles around, the AI needs to recognize few objects and its processing time is lower than the case that AI is

on a downtown and many vehicle are crossing in front. For my simulation environment's case, an ad hoc basis task scheduling will be done, looking at the resource utilization (CPU, memory, network bandwidth etc) and relocating VMs to remove the lack of resources.

The unique feature of my simulation environment is the spontaneous movement of VMs. According to the explanation with figure 4.2, if a self-driving vehicle moves over the boundary of the area. the VM represents its self-driving vehicle also moves to another physical machine. This deployment mechanism can utilize "geographical locality" of data transaction between VMs and simulators, in contrast, spontaneous movement of VMs occurs. Load balancing mechanism can not control these spontaneous movement. This feature will be "the third reason" which causes the load imbalance and has not ever researched in the previous works.

5.2.2 Virtual Machine based Simulation Environment

There are various works of virtual machine based simulation environment in network testbed research. [54], [29], [8], [58], [43] are the examples of such simulation environment. These environments is used mainly for performance test of a network application. They basically assumes that each VM has enough resource to keep real-time processing, so there is no processing delay caused by task scheduling for VMs. In other words, none of them deal with time synchronization between VMs. Yoginath et al introduces time synchronization mechanism between VMs is in literature [56], but it is effective within a physical machine which executes several VMs.

5.3 Solution

5.3.1 Basic Concept

We can not avoid the lack of CPU against the spontaneous increase of VMs in an area, so it should be a feasible solution to choose a minimum set of VMs which we should set high priority to avoid the lack of resources. An objective of my simulation environment is to offer a feasible test for multiple self-driving AI, so there are a few VMs which are executing self-driving AI we focus on, and If we can choose a set of VMs which have interaction with these self-driving AIs, we can also ignore the other VM's synchronization. "Interaction" indicates a situation that a vehicle which is visible on the other vehicle's sensor. In a case of camera, if a vehicle is visible on the other vehicle's camera, the latter vehicle interacts the former vehicle in the point of object recognition.

Figure 5.2 describes the grouping image of vehicles on a simulation. There are five vehicles (Vehicle A to E) in this figure and they have direction to go (depicted by arrow). VehicleA is going down on a road and VehicleB is going up on the same road. Finally, they will pass by each other somewhere on this road. If they have camera for object recognition, then this camera must capture the opposite vehicle (A camera on VehicleA captures VehicleB, vise versa). Same as VehicleA and B, VehicleC, D and E will pass by each other on the right side of this area, so they are captured by their own camera. I name VehicleA and B as "GroupA" and VehicleC, D and E as "GroupB". In this figure, vehicles belong to GroupA have never encountered the vehicles belong to GroupB, therefore, we can set different priority to each group. If we have to observe the behavior of Vehicle A, we can set high priority to the vehicles belong to GroupA, in contrast we must ignore the behavior of vehicles belong to GroupB.

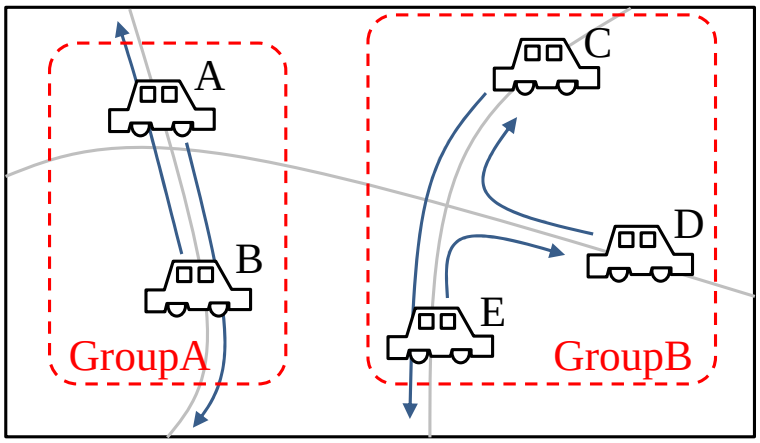

Figure 5.2: Basic concept

Looking at the Figure 5.2, there are two groups and they completely independent from each other. However this is a rare case that several independent groups are made in a vehicular simulation, indeed a part of the vehicles in a group also belong to the other groups. There would be a situation that from VehicleA to E have already gone to out of the map of Figure 5.2, and a new vehicle (VehicleF) is coming into the map. In this situation, if VehicleF went through the horizontal road depicted on Figure 5.2, VehicleF would belong to the both GroupA and GroupB, although VehicleF has never encountered to the other vehicles. Therefore, I must deal with "elapsed time" when I make an interaction group.

Figure 5.3 indicates the relationship between elapsed time and vehicle's position. The left rectangle indicates the positions of two vehicles on a certain clock (t=0), and the right is another clock (t=1). The two vehicles are about to interact somewhere on a road at t=0 and then they are going to their own direction at t=1. Until the two vehicle pass by each other, they belong to the same group and I must put the same priority to

them. However, after that they have already interacted and are going to different direction, they are independent from each other and I do not put same priority to them anymore. To acquire the relation between elapsed time and vehicle's position, I must predict the interactions before a self-driving simulation.

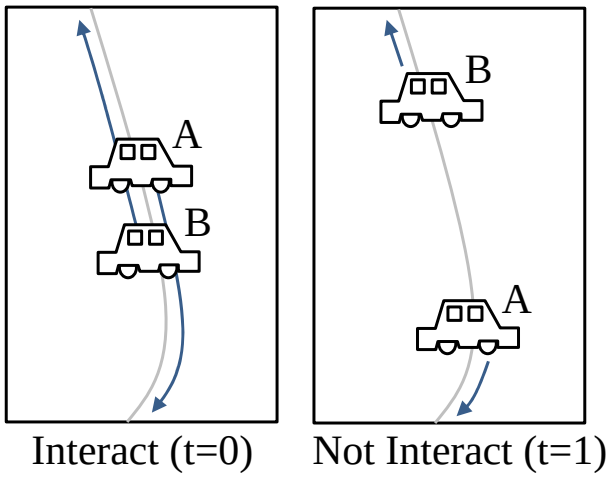

Figure 5.3: Vehicle's position on each time period

In the following sections, I would like to discuss the definition of interaction, prediction of interaction and implementation of an algorithm to make the interaction groups and show a control flow to set priority to the VMs.

5.3.2 Definition of Interaction

As I said it in the previous subsection, "interaction" indicates a situation that a vehicle is visible on the other vehicle's sensor. There are many kinds of sensors, camera, LiDAR(Laser Imaging Detection and Ranging), sonar(sound navigation and ranging), and so on. The common feature of these sensors is the fact that they have "effective range" and individual shape. For example, camera has range with dozens of meters and it shapes triangle in front (and angle is different from each other), LiDAR has several meters range

of circle. If a vehicle comes in a range of sensor, self-driving AI recognizes it as an objects and then make some decisions: ignore because it is not on their way, avoid not to hit, etc. Therefore, I define interaction as a situation that a vehicle comes in an effective range of sensor and it interacts a self-driving AI's decision. My simulation environment must get the effective range of sensors for making the above groups.

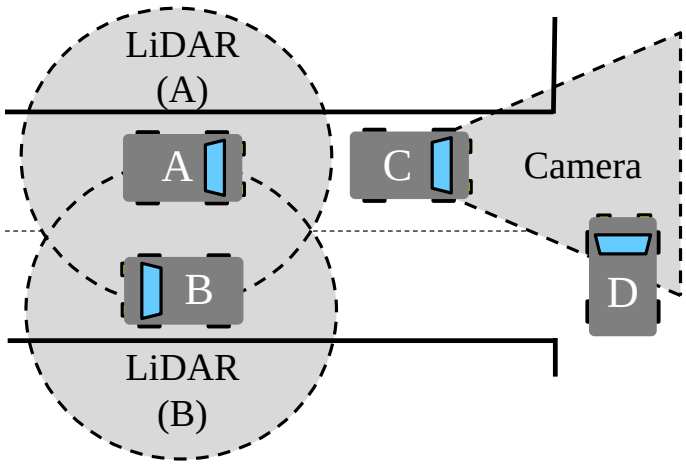

Figure 5.4: Vehicle's position at a time point and the scopes of sensors

Figure 5.4 is an example to explain the interaction. There are four vehicles (A to D) and VehicleA and D is going to East, VehicleB is going to West, and VehicleD is going to North. VehicleA and B have their own LiDAR and VehicleC has a Camera. Their range is depicted with short dashes line (circle of LiDAR, triangle of camera). In this situation, A self-driving AI on VehicleA recognizes that VehicleB exists in the range, but VehicleA would not do anything because VehicleB runs opposite side and it does not interrupt the way. Strictly speaking, VehicleB does not interact the decision of self-driving AI on VehicleA, but I regard this case interacts a self-driving AI's decision on VehicleA because we can not know what it thinks. VehicleB is same as the situation of VehicleA. VehicleD

is about to get across in front of VehicleC. About the case of VehicleC and D, VehicleD obviously interacts the decision of VehicleC, but VehicleC does not interact the decision of VehicleD because it does not have any sensors.

According to the above discussion, there are two cases about interaction information. One is "unidirectional interaction" and another is "mutual interaction". Unidirectional interaction and mutual interaction must be distinguished in the Grouping phase. I use the following symbols on the later section.

1. A $->$ B (VehicleA interacts VehicleB)
2. $A <=> B$ (VehicleA and B have mutual interaction)

In the figure 5.4. There are two interaction information that VehicleA and VehicleB have mutual interaction($A <=> B$), and VehicleC has one way interaction with VehicleD ($C -> D$). Here I consider a case that I focus on VehicleC's real-time processing, there is one way interaction information ($C -> D$), so VehicleC and VehicleD must be in a same group and set high priority. On the other hand, when I consider VehicleD's real-time processing, I set high priority only to VehicleC because VehicleD do not recognize VehicleC. In the figure 5.4, A camera device attached on VehicleC watches VehicleD is going to cross over in front of VehicleC, so VehicleC will recognize VehicleD. In contrast with VehicleC, A camera device attached on VehicleD do not watch any vehicles in front. Therefore, One way interaction information ($C -> D$) is given in this case.

It is controversial whether or not effective range of a sensor is enough as interaction information. For example, There is a typical case of traffic accident that a vehicle comes from diagonally behind which is the outside scope of a sensor, and then two vehicles have

a collision. For the time being, My simulation environment can not deal with the above case because it calculates interaction information from the result whether or not a vehicle is visible on a sensor. If any sensors does not catch a vehicle comes nearby, my simulation environment does not grantee real-time processing of such vehicle for the time being. I would like to discuss how to deal with these cases in the later section.

5.3.3 Prediction of Interaction

I need both the definition of interaction and the positions of vehicles at every time point to predict the interactions. I can not acquire the positions of vehicles at every time point before finishing a self-driving simulation because a self-driving simulation creates the positions of vehicles. Hence, I try to understand the interact information with the preliminary traffic simulation. before a self-driving simulation is executed. I used SUMO [34] traffic simulator to acquire the positions of vehicles at every time point.

As the inputs to SUMO, current position and route information of each vehicles and a map are required. These information and the map can be acquired from Simulation Middleware I prepared. As I said it in the Figure 3.1, every VM and simulator are connected via Simulation Middleware and every information is stored in a database that the Simulation Middleware has.

I assume that current position and route information can be acquired from somewhere in my simulation environment. I can get "current position" from the simulation middleware I explained with the figure 3.1 at least. Route information which road a vehicle goes through should be acquired from each VM. Self-driving AI must need to set destination and route information before departure, so these information must be existed whenever I may run

the preliminary traffic simulation. I also assume that destination or route information can be acquire from each VM through a certain API.

5.3.4 Grouping

Grouping is to make a group which interacts to a decision of self-driving AI on a vehicle. As I mentioned about the meaning of group in the Basic Concept section, vehicles in a group interact each other through the sensors which are attached on them.

Figure 5.5 is a time sequence which is used for searching interact information and grouping. Each circle with a capital letter means vehicle on a simulation and horizontal axis indicates the elapsed time. An arrow in this figure means a vehicle moves to another place, so VehicleA on T=0 has different position from VehicleA on T=1. Ovals indicate interact. Red oval includes two case of interact, one is unidirectional interact, another is mutual interact. For example, VehicleC and VehicleD has interacts on T=1 and T=5 and VehicleA, B and C have interact information on T=3. short dashes line surrounds almost half part of the figure indicates the group which I finally want to make. This figure describes the group of VehicleA. The group of VehicleA means "VehicleA and the other vehicles which interact VehicleA". Looking at whole simulation time (T=0 to T=5), none of independent group can be made if I make it based on a concept that one vehicle should belong to only one group.

I also show an example algorithm of grouping in Algorithm 5. There are three input data : interact information (described as influInfo in this algorithm), a target vehicle, and simulation end time of a vehicle. Output is a tuple list of time and the other vehicle which interacts the target vehicle.

Algorithm 5 Grouping

Require: $influInfo$: *interact information of all vehicle*
Require: $vehicle$: *identification of a vehicle*
Require: $time$: *simulation end time of a vehicle*

```
1: function INTERACT(vehicle, info)
2:     (v1, v2, symbol) ⇐ info
3:     if v1 == vehicle & symbol == "<=>" then
4:         return v2
5:     end if
6:     if v2 == vehicle & symbol == "<=>" then
7:         return v1
8:     end if
9:     if v2 == vehicle & symbol == " -> " then
10:        return v1
11:    end if
12:    return None
13: end function

14: function SEARCH(influInfo, vehicle, time)
15:    retlist ⇐ [ ]
16:    while time >= 0 do
17:        inflist ⇐ influInfo[time]
18:        for each inf ∈ inflist do
19:            v ⇐ interact(vehicle, inf)
20:            if v! = None then
21:                retlist + [((v, time)]
22:            end if
23:        end for
24:        time − −
25:    end while
26:    cutRedundancy(retlist)
27:    return retlist
28: end function

29: function GROUP(influInfo, vehicle, time)
30:    retlist ⇐ SEARCH(influInfo, vehicle, time)
31:    if retlist == [ ] then
32:        return retlist
33:    else
34:        for each ret ∈ retlist do
35:            (v, t) ⇐ ret
36:            return retlist + GROUP(influInfo, v, t)
37:        end for
38:    end if
39: end function
```

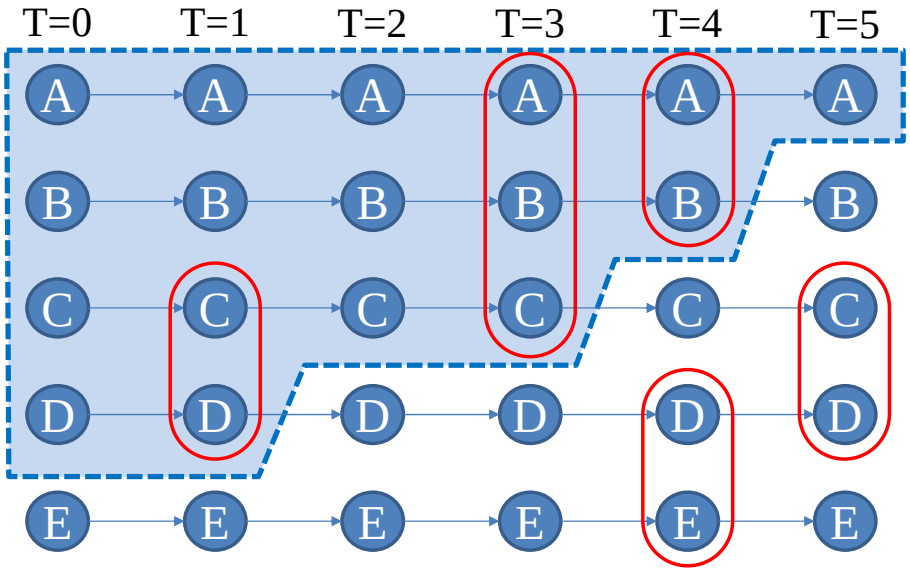

Figure 5.5: interaction search and grouping

5.3.5 Control Flow (set high priority to each vehicle)

Figure 5.6 describes control flow of my simulation environment. There is the simulation middleware which gives priority controls to two physical machines in this figure. This simulation middleware is composed of four components: Traffic Simulation, interact Search, Grouping and Priority Control.

First of all, when a vehicle is appeared on a simulation, a virtual machine which represents its vehicle is executed on an appropriate physical machine. Before the vehicle start running, I assume that a route to a destination is given as I said it on the section 5.3.3. The simulation middleware receives the route information via some procedures. One option will be a car navigation system that my simulation environment prepares on a virtual machine.

When the simulation middleware receives the route information from all vehicle appeared on a simulation, it executes traffic simulation to calculate the vehicle's state (place, direction, speed etc) as a "prediction". With the simulation result, the simulation middleware executes interaction search I explained it on the section 5.3.2. Interaction search creates interaction information that is composed of a set of vehicles which is visible on each other's sensor and clock when they are visible. Red ovals on the figure 5.5 is the example of interaction information. And the next, the middleware executes the grouping. Finally the middleware send the priority information to each physical machine and the priority information is set to the designated virtual machines.

The most important thing is the fact that one vehicle that an owner of a simulation would like to test is needed to make a group of vehicles because the essential idea of my approach is to abandon the real-time processing of all vehicles appeared on a simulation, instead of saving the real-time processing of the critical vehicles that interact with the target vehicle. If an owner of a simulation focuses on one vehicle, for example VehicleA in the figure 5.5, the grouping component makes a group of vehicles that have some interaction with VehicleA. VehicleE is executed with the lower priority and from T=2, VehicleD's high priority property is cancelled.

The simulation middleware executes this control flow every time when one of the vehicles in a simulation changes its route. Such situation will happen when a vehicle acquired a road construction information on the way to a destination and enable to continue the navigation. In this situation, the simulation middleware calculates all vehicles state again because a route change of one vehicle could be a trigger to change the interaction information of all vehicles. "Butterfly effect" would be a similar expression. A route change

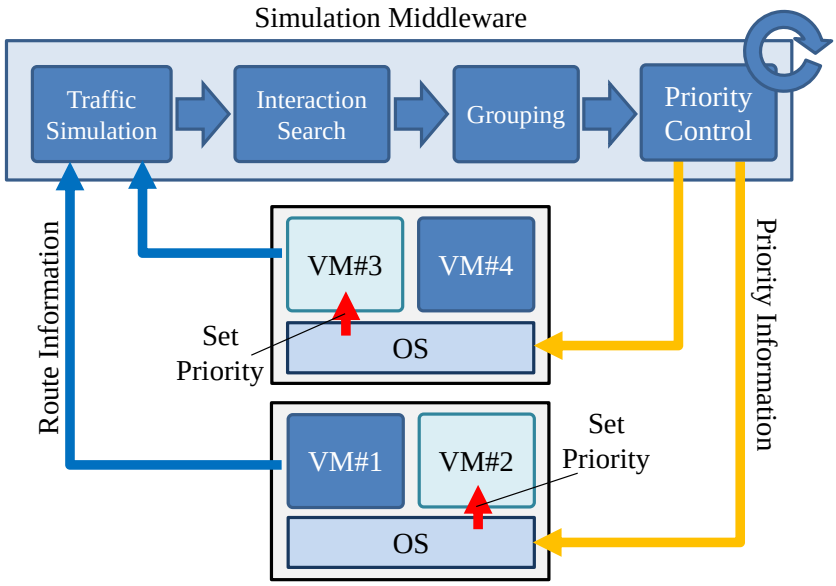

Figure 5.6: Priority Control Flow in my simulation environment

will directly influence to a vehicle which will encounter the vehicle changed its route, but such vehicle also gives some effects to the other vehicles. Referring at the figure 5.5, when Vehicle B change the route on T=2, it directly influences to the interaction information of VehicleA and VehicleC, but it also influences to the interaction information of VehicleD on T=5. Interaction basically requests some action to the related vehicles. It will be imagined as an interaction that one vehicle is crossing over an intersection and another vehicle is also about to enter its intersection but it temporarily stops to avoid an accident. In this case the latter vehicle is forced to take an action (braking) and this action takes additional time to cross over the intersection. Now, I assume that VehicleC is in similar situation when it encounters VehicleB and VehicleA on T=3. If this interaction was canceled because of the route change of VehicleB, VehicleC would not stop temporarily and reach the next interaction that it encounters VehicleD on T=5, faster than before. As the result, VehicleC would not have interaction with VehicleD on T=5 because VehicleC arrives earlier than

VehicleD does. VehicleD also gives some effects to the other vehicles on the next interaction. Consequently, I think that my simulation environment must calculate the interaction information of all vehicles again.

5.4 Evaluation

In this section, I describes the evaluation result. This section is composed of three subsection. I explain the basic sequence of the evaluation at first. And then, I discuss the evaluation points. Finally, I show the result.

5.4.1 Evaluation Environment

Figure 5.7 is the basic sequence of the evaluation environment. The basic sequence is composed of a map and route information of every vehicle as input, simulation middleware, VM deploy algorithm, a cloud simulator. The simulation middleware is also composed of a traffic simulator and the other parts. I prepared SUMO traffic simulator to predict the vehicle's position and also created the three components, interaction search, grouping and priority setting. The simulation middleware creates the priority setting of each vehicle and input them into the cloud simulator. I made a cloud simulator, instead of using an actual cloud because of the following two reasons. First reason is that I need to evaluate the ideal ability to keep real-time processing of my approach. If I used an actual cloud and management tool (ex. OpenStack), the actual cloud and management tool would influence the capability of my approach. Second reason is that I can not prepare more than several hundreds of physical machines to evaluate my approach because the preparation takes a lot of time. To output the processing time of each job and investigate the real-time property of my approach, a cloud simulator is feasible for this purpose, compared with the preparation

of an actual cloud environment. The cloud simulator also needs the deployment information of all virtual machines. To create the deployment information, I used VM deploy algorithm that I proposed on the chapter 4.

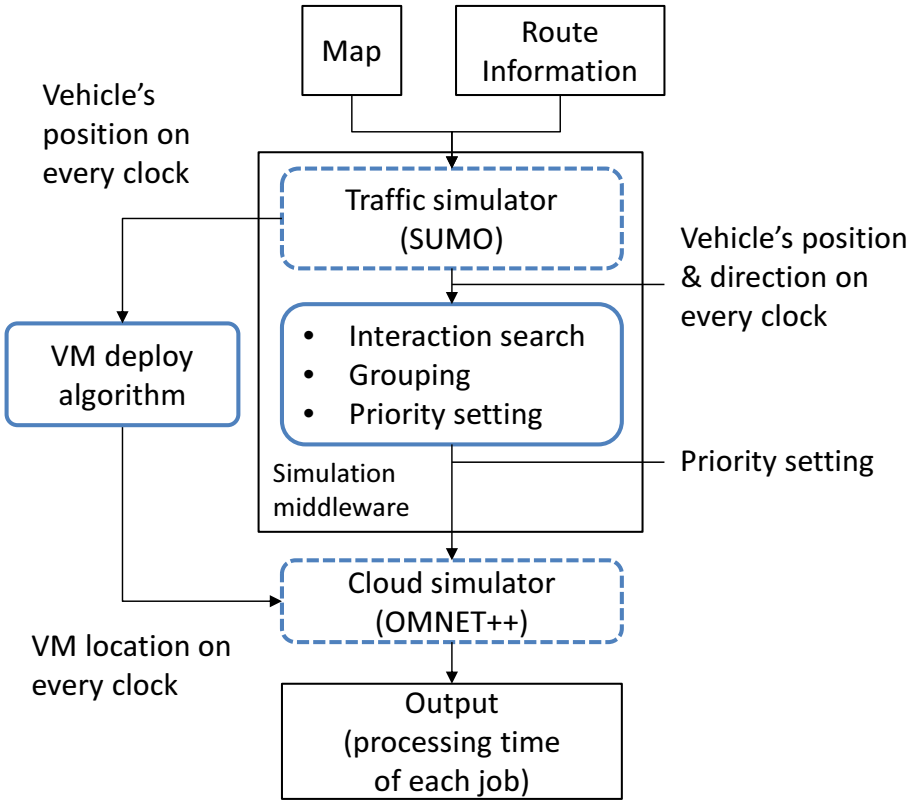

Figure 5.7: Basic sequence of Evaluation

I also describes the detail of the cloud simulator in the figure 5.8. The cloud simulator is based on a queuing system. A physical machine is composed of one server (depicted on the figure as CPU), one sink, two queues and one selector. A number of CPUs will be changed in accordance with how many CPU is attached on one physical machine. A number of physical machine is also changed in accordance with how many physical machines are used for this simulation. The sources in this figure represent the vehicles. The vehicles in this

figure are equal to self-driving AIs running on the virtual machines and these AIs produce several kind of jobs. One of them would be image processing and object recognition. Another would be a graph search processing to find an appropriate route. I assume that most of the jobs produced by self-driving AI is a constant interval task. A source in this figure also create a job with constant interval and a physical machine processes the job with a constant time. Properly speaking, virtual machines which represent the vehicles should be inserted between the physical machines and sources in the figure 5.8, but I omitted the virtual machine layer because most of the CPU time is consumed by the processing of the job and the cost of virtual machine layer is enough small to omit.

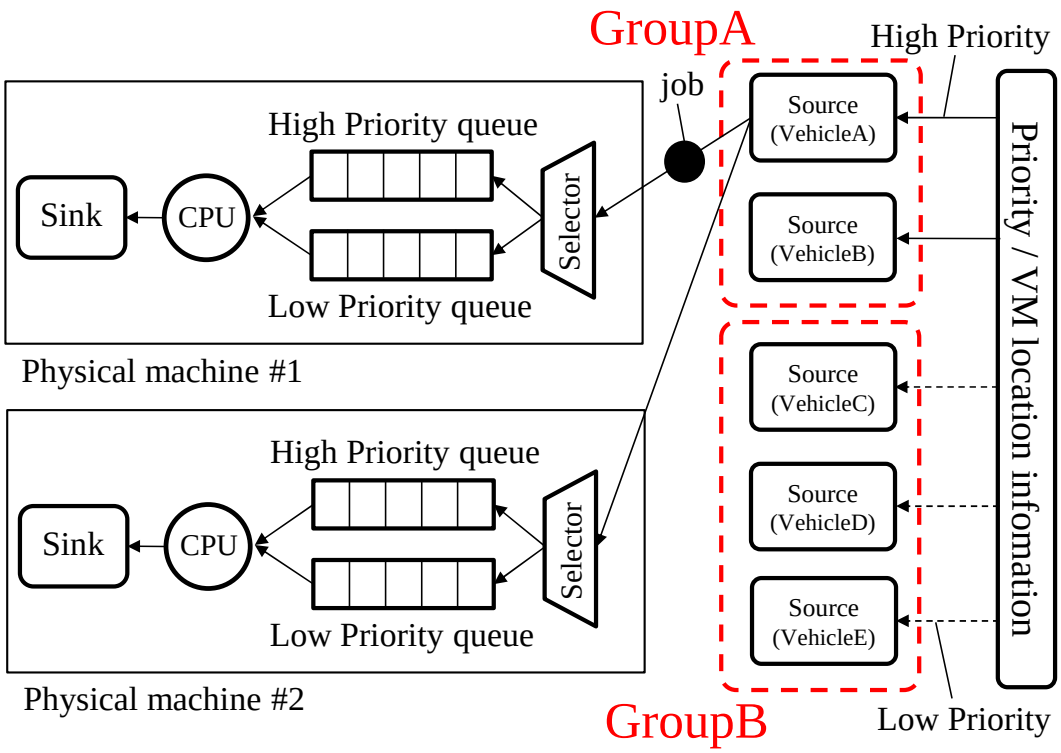

Figure 5.8: Cloud simulator (OMNET++)

A source receives priority information and VM location information. A priority information is a set of time and priority. The time in the priority information is the finishing time of the priority. For example, when a source receives a set of time and priority like (4.0, high), the source changes the job's priority to high immediately and it will be finished by 4.0 clock. a source also receives VM location information. VM location information is a list of locations. Each location is a set of a time and the id of a physical machine. The time indicates to pick up a next location from the location list. When a source receives VM location information, it picks up one location and set the destination of a job. When the simulation clock reaches the time, a next location is picked up and set the destination to a job.

Table 5.1 is the configuration that is used for a traffic simulation in the simulation middleware. SUMO traffic simulator needs a map and route information (appealing place and route) of each vehicle as input. I made a grid-shaped load map and vehicle's information with the scripts which is a part of SUMO simulator. Vehicles are generated in the range from 100 to maximum 500 vehicles at the beginning of the simulation. I prepared two method to make a vehicle's route. One is based on random walk. Another is the route that compels a vehicle to go through a designated place. Lesser interaction is better for reducing the number of vehicles which are given high priority. Random walk is an ideal situation to reduce the interactions. Compelling transit via some places is for creating a traffic jam in those places. In contrast with random walk, a traffic jam is the worst situation in the point of reducing the interaction. If a sensor attached on a vehicle catches another vehicle, they have an interaction. A traffic jam makes a long line of vehicles, so a vehicle have at least two interactions with anteroposterior vehicles in the case of 360 degree sensors (WiFi, LiDAR etc). Consequently, a sort of "interaction chain" is made. If

the interaction chain includes the target vehicle that I are planning to test, all vehicles in the interaction chain should have high priority. I think that to evaluate the both ideal and worst case for assign high priority is important to know the effective range of my approach.

Table 5.1: Traffic simulation parameters

map	$2km \times 2km$ square, 400 junctions, two-way loads
number of vehicles	100 to 1000 vehicles
simulation time	200 to 1000 seconds
each vehicles emersion	beginning of the simulation
route of each vehicles	Random or partially designated route

5.4.2 Evaluation Points

I consider the following 5 points to evaluate my approach. The objective of my approach in this chapter is to make a minimum group of vehicles those I am planning to test, and set high priority to them to keep real-time processing. The smaller number of vehicles in one group is better. If the number of vehicles was getting bigger, it would be difficult to keep real-time processing because the number of VMs that should assign high priority property would exceed the number of CPUs. Consequently, and finally a lack of CPU time would occur. The following items are for investigating what circumstance is effective for my approach and what is ineffective. I would like to show the result in the next subsection.

1. measure the processing time of jobs with the fixed number of vehicles and the fixed duration of a simulation as a fundamental evaluation, and then compare the two circumstances; my approach which assign high priority to a group of vehicles and a normal situation which does not assign a priority property.

2. measure the processing time with my approach in the case that the number of vehicles in a simulation is increased

3. measure the processing time with my approach in the case that the duration of a simulation is increased

4. investigate difference in the number of interaction between the vehicles running with random walk and the vehicles stay under a traffic jam

5. investigate difference in the number of interaction related to difference of sensor scope

5.4.3 Result

Fundamental evaluation

Figure 5.9 indicates the fundamental evaluation result of my approach. I also describe the configuration of this evaluation in the table 5.2. I made the cluster of 9 physical machines. Each physical machine linked to a part of the map which is used for traffic simulation. The basic idea is mentioned on the section 4.2. A vehicle on a map is given as a virtual machine and it is executed on a physical machine which a certain part of a map is assigned (refer to the figure 4.2). I proposed Voronoi decomposition and introduced dynamic assignment of segments to equalize the number of virtual machine in a physical machine. However, I did not use the Voronoi decomposition in this evaluation to see the effect of priority setting. Instead of Voronoi decomposition, a map used for traffic simulation is separated into 9 parts equally and a physical machine is assigned to each segment. As the result, unbalancing of the number of virtual machines on one physical machine occurs in this evaluation. This unbalancing of the number of VMs is equal to the unbalancing of the number of jobs in one physical machine and the processing of a job is delayed on a physical machine which has large amount of jobs caused by the unbalancing.

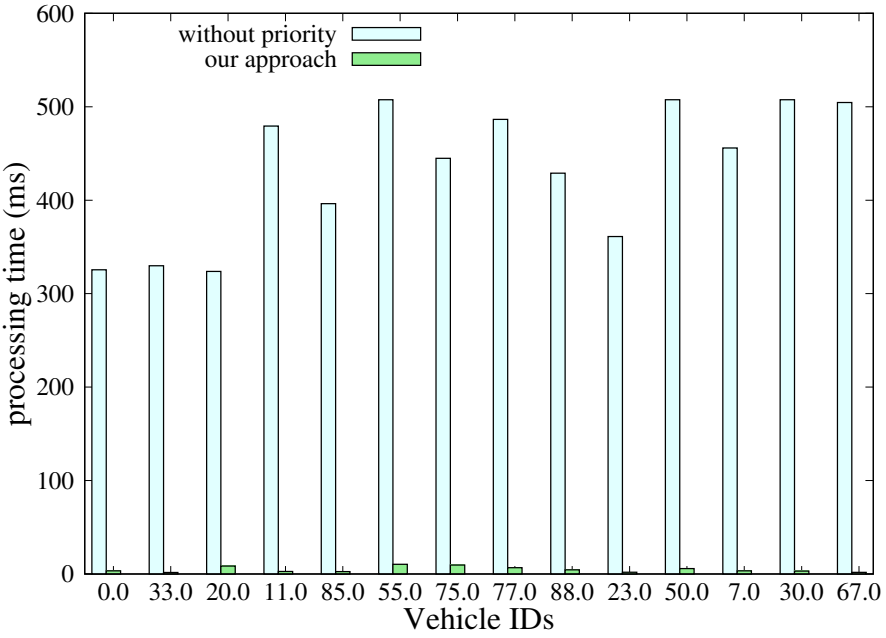

Figure 5.9: Fundadamental evaluation of my approach

Figure 5.9 revealed the fact that my approach could effectively reduce the processing time, compared with the processing time without my approach. The x axis indicated the individual vehicle ID and y axis indicated the average of processing time of jobs produced by each vehicle.The best case was the vehicle ID "30.0". The processing time of my approach was 3.03 milliseconds and the processing time without my approach was 507.53 milliseconds, it means my approach reduce the processing time to about one hundred sixtieth. The worst case was the vehicle ID "55.0". In the case of the vehicle ID "55.0", my approach could reduce the processing time to about fiftieth, compared with the processing time without my approach. An average processing time with my approach was 4.703 milliseconds, and the one without my approach was 432.808, therefore my approach reduces the processing time to about one ninetieth.

Table 5.2: Configuration of the fundamental evaluation

Traffic simulation configuration		
	map	$2km \times 2km$ square, 400 junctions, two-way loads
	number of vehicles	100 vehicles
	simulation time	200 seconds
	route of each vehicles	Random walk
Job properties		
	processing time	2 milliseconds
	generate interval	2 milliseconds
physical macine configuration		
	the number of physical macines	9
	the number of CPUs	4 core
interaction search configuration		
	sensor range	transparent circle(20 meter radius)

The jobs produced by vehicles takes 2 millisecond to be processed, therefore, if they smoothly processed without waiting on a queue, the ideal result of the above evaluation would be 2 milliseconds. An average processing time with my approach was 4.703 milliseconds. This means the fact that some jobs were slightly stacked on a physical machine. I discuss whether or not this delay is reasonable for the real-time processing of self-driving simulation later.

Influences on the increasing the number of vehicle

Figure 5.10 indicates the relationship between the number of vehicles and processing time. The configuration of this evaluation is same as the fundamental evaluation. The x axis means the number of vehicles in a simulation and y axis means the average of processing time at all vehicles. As the number of vehicles in a simulation increase, the average of processing time also increase. Under 700 vehicles, the average increases larger than the above of 700 vehicles. I think that $2km \times 2km$ square map is too small for more than 700

vehicles. In fact, I observed there are many places where some vehicles are stacked. Under 300 vehicles, the average of processing time was almost flat, compared with the other parts in this graph, therefore, to keep real-time processing with my approach, I should execute the self-driving simulation within 300 vehicles in the case that a map size was $2km \times 2km$ square.

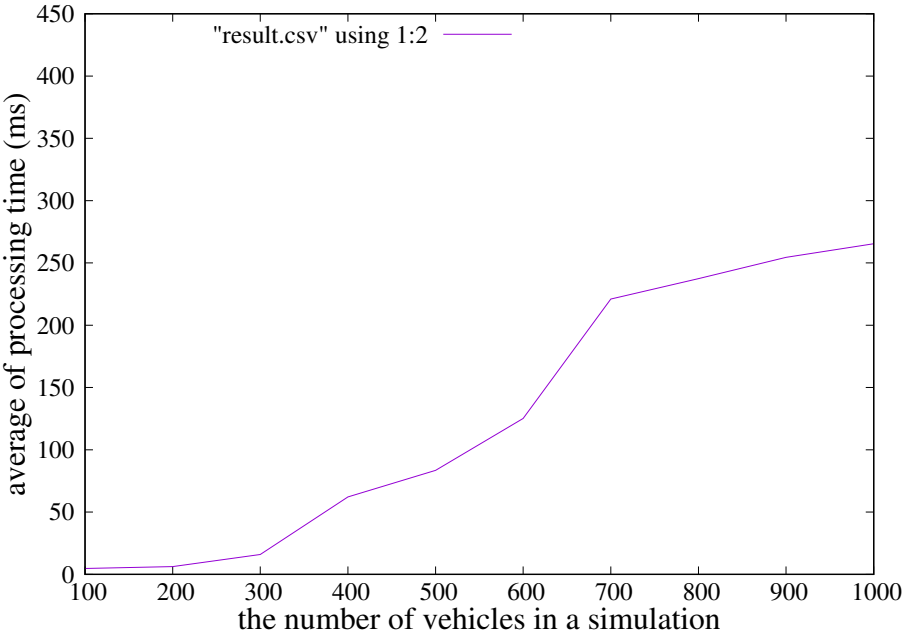

Figure 5.10: Influences on the increasing the number of vehicle

Influences on the increasing the duration of a simulation

Figure 5.11 indicates the relationship between the duration time of a simulation and processing time. The configuration of this evaluation is same as the fundamental evaluation. Compared with the figure 5.10, the average of the processing time is slowly increasing as the duration of a simulation become longer. This result also indicates the fact that my approach can not keep the real-time processing in a simulation which has longer duration.

Looking at the 400 seconds duration in the graph, the average of processing time is about 50 milliseconds. As I mentioned the reasonable processing time in the section 3.7, the delay can hardly be noticed up to 50 millisecond and acceptable up to 100 millisecond, if no high demands with respect to realism are needed. Therefore, the duration of a simulation should be less than 600 seconds in this configuration. I will discuss the matter how to take the duration longer exceed 600 seconds in the later section.

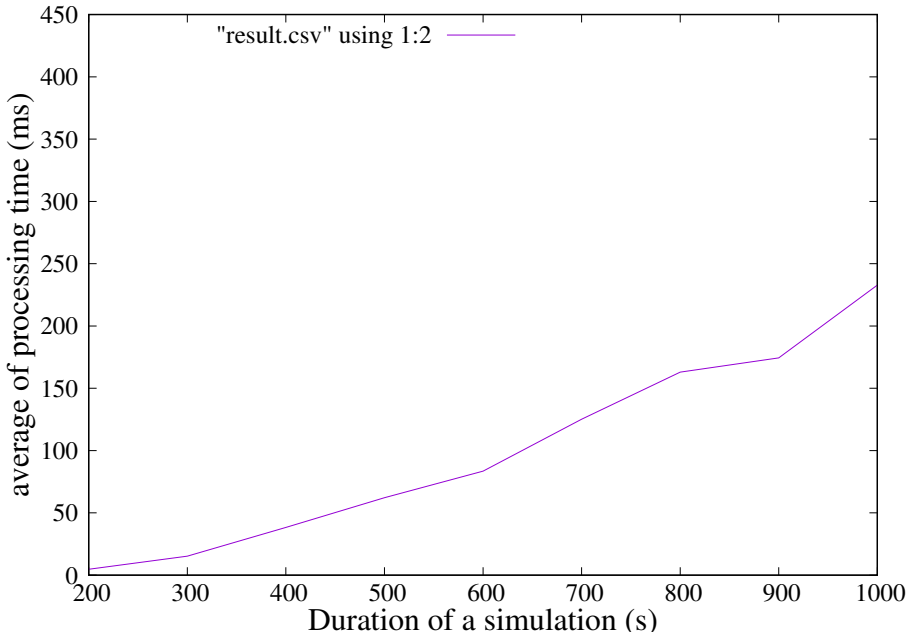

Figure 5.11: Influences on the increasing the duration of a simulation

Differences of the number of interactions between random walk and traffic jam

Figure 5.12 indicates how the number of interactions vary in the case of vehicles running with random walk and vehicles make a long line because of a traffic jam. I prepared three cases that has different number of vehicles and compared random walk and traffic jam. The other configuration (duration, map, etc) is the same as one used in the fundamental

evaluation. The number of interactions in the case of random walk is almost same as the evaluation result described in the figure 5.10. On the other hands, the number of interactions in the case of traffic jam is drastically increased. This result indicates the fact that almost all vehicles have interaction each other if they make traffic jam in a simulation, hence, traffic pattern should be made carefully not to induce a traffic jam. I put an assumption that a sensor attached on a vehicle has the range of transparent circle and the radius is 20 meter (described on the table 5.2).

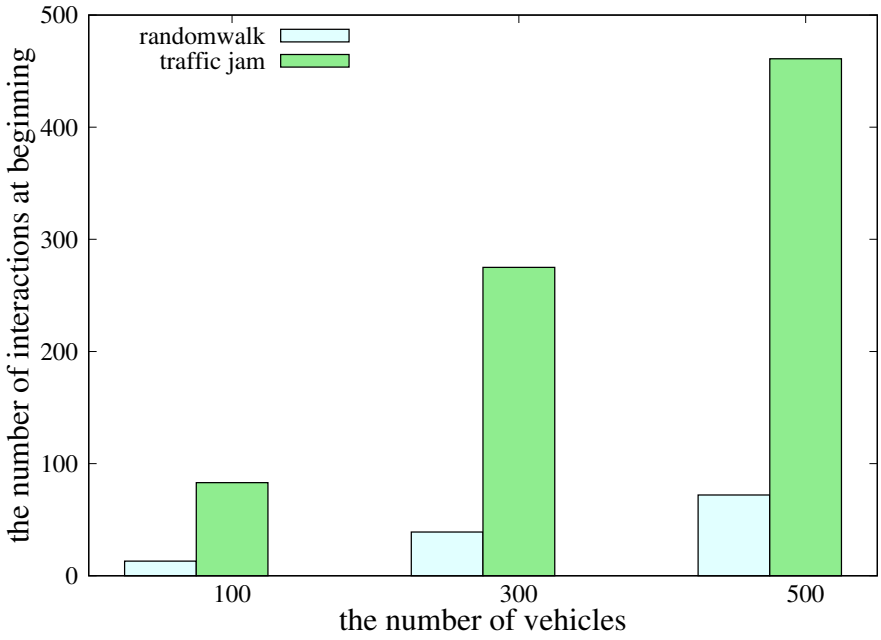

Figure 5.12: Differences of the number of interactions between random walk and traffic jam

Differences of the number of interactions between several sensor scopes

Figure 5.13 indicates how the number of interactions vary if the sensor scope is different each other. I prepared four types of sensor scopes; transparent circle, transparent sector,

non transparent circle, non transparent sector. A representative of sensors which have the scope shaped transparent circle is WiFi. The case of transparent sector is a beam forming microwave, the case of non transparent circle is a LiDAR, and non transparent circle is a camera. I changed the number of the vehicles in a simulation to 500 and also changed routing pattern of each vehicle from random walk to traffic jam. Because I need to emphasize the gap of the number of interactions between the different kind of sensors. Looking at the result, the number of interactions in the case of transparent circle is almost 500. In contrast with it, the case of non transparent sector is about 100. The number of interactions vary in accordance with the range, angle of view and transparency. A sensor that has the scope shaped transparent circle is the worst case in the point of interaction search because it has wide range, 360 degree angle of view, and transparent through an object, therefore, this sensor catches the anteroposterior vehicles and it makes interaction chain which I mentioned in the section 5.4.1. In the same way as a traffic jam in a simulation should be avoided, I must give attention not to attach any sensors which is not used during a simulation.

5.5 Discussion

5.5.1 Evaluation Summary

I summarize several characteristics that my approach have in accordance with the evaluation results in the previous section.

- Under the ideal circumstance for a self-driving simulation, my approach can reduce the processing time to about one ninetieth.
- I should execute the self-driving simulation within 300 vehicles in this configuration.

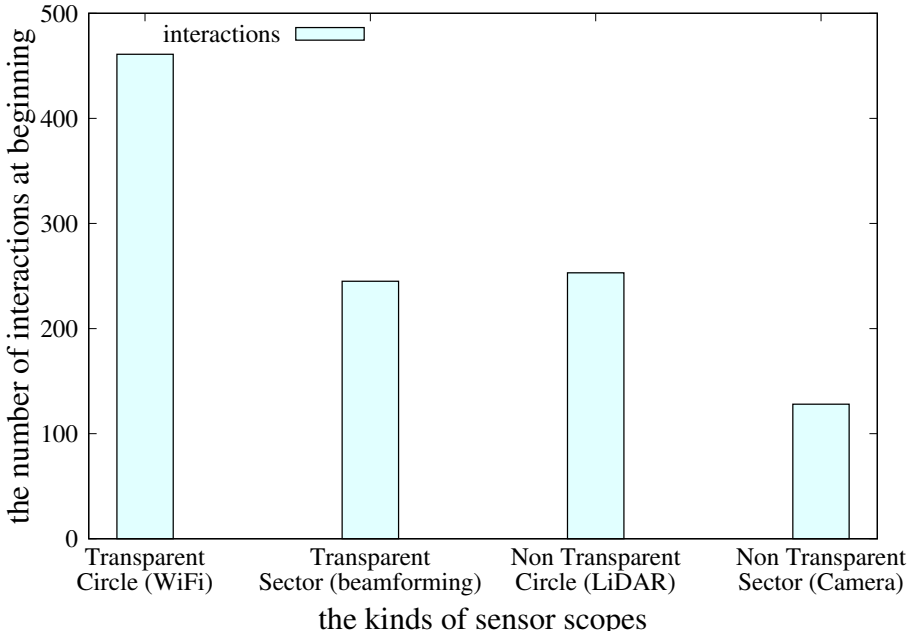

Figure 5.13: Differences of the number of interactions between several sensor scopes

- The duration of a simulation should be less than 600 seconds in this configuration.
- I do not use it under a traffic jam.

The acceptable number of vehicles in a simulation depends on the map size. I used $2km \times 2km$ map size. If the map was bigger, it would be acceptable to execute more than 300 vehicles. In other word, **the number of interactions depends on the balance of the number of vehicles, duration and map size**. Moreover, the acceptable number of interactions depends on the number of CPUs in a segment. I suggested a sort of "criterion" through this evaluation, when my approach is used in a self-driving simulation for working appropriately to keep the real-time processing.

5.5.2 Consider an Operation Under the Long Duration

There are some cases that an user of a simulation would like to test its self-driving system with long duration. The evaluation result of my approach indicates the fact that the duration of a simulation should be less than 600 seconds in this configuration. When the 600 seconds simulation finished, there would be "clock distortion"between the two groups of vehicles. The vehicles which belong to high priority group will keep the real-time processing or the clock distortion between them less than 100 milliseconds, according to the result in the figure 5.11. The vehicles which belong to low priority group may not keep the real-time processing and the clock distortion between the vehicles in high priority group and low priority group become more than 100 milliseconds and more than 100 milliseconds delay is clearly observable and the simulation environment can not keep a realistic simulation.

For the time being, unfortunately I have no choice but to run the prediction loop again at the time when the simulation reached a certain seconds. If an user wants to keep the delay within 100 milliseconds, my simulation environment must run the prediction loop every 600 seconds, and also the user must accept the circumstance that there are time distortion between some vehicles. And also, the vehicles in low priority group does not reach the place where they should be when the simulation clock is 600 second. In this case, my simulation environment must run the prediction loop again with the current places and routes. It would be the so-called "reset".

5.5.3 Consider the Combined Usage of Voronoi Decomposition and Priority Setting

I consider the next step to a novel approach that combines the Voronoi decomposition based deployment I suggested in the chapter 4 and interaction prediction based priority setting suggested in this chapter. The priority control mechanism suggested in this section includes a prediction, on the other hand, the Voronoi decomposition based deployment mechanism does not use any prediction, hence, if the deployment mechanism used the prediction information which acquired from the priority control, the amount of physical machines that the deployment mechanism use would be reduced. The deployment mechanism add a physical machine when the number of VMs on a physical machine exceed the number of CPUs on that physical machine. For example, I assume the situation that there is one physical machine which have 10 cores and also there are 10 VMs on the physical machine, and one additional VM is about to be moved from another physical machine. In this situation, if the five of 10 VMs have high priority and the others are low, the deploy mechanism do not need to add an additional physical machine because the physical machine can still keep the real-time processing of the vehicles which have high priority setting. Consequently, the number of physical machine that the deploy mechanism use will be reduced. I am going to consider how to combine them as the next step of the cloud load balancing mechanism.

5.6 Conclusion of this chapter

I suggested a novel approach which enables to assign priority information in accordance with the interaction information during the self-driving simulation. My approach can reduce the processing time of the jobs those are created by the self-driving AIs to one nineti-

eth. This result indicates the fact that my approach contribute the real-time processing during CPU overload.

Chapter 6

The other topics for a massive distributed simulation

In this chapter, I describe the other topics related to create a massive distributed simulation. This chapter is composed of two sections.

One is the topics related to time synchronization. My simulation environment uses virtual machine technology as an agent and there are no time synchronization mechanism to keep the inside of VMs independent from my simulation environment. Instead of a time synchronization mechanism, I prepared a load balancing algorithm in the chapter 4, and a context aware scheduling algorithm in the chapter 5 that keep real-time processing. Strictly speaking, my simulation environment needs a time synchronization mechanism in the case that two algorithm could not keep real-time processing and the clock gaps between agents were getting bigger. Therefore I refer the existing time synchronization mechanism in this section.

Another is the topics related to connectivity with the other simulation environment. Simulation technology is used at various field and there are many kinds of simulation technology. Basically my simulation environment connect the simulators with a database, so if we would like to connect the other simulators to my simulation environment, this simulator should use the database and store the data they created with a certain format. I mainly discuss the time gap compensation when we would like to connect a new simulator to my simulation environment.

6.1 Time Synchronization

Literature [21] describes some research topics related to parallel and distributed manner of discrete event simulation and there are several works for making scalable discrete event simulation environment [38] [17] [25]. One of the main topics is time synchronization of each agent. An agent simulation assume the fact that each agent is executed simultaneously and all of the clocks which each agent has synchronized. For example, if there was "time skew" between each agent during traffic simulation, it would happen that the simulation indicates no traffic jam on a crowded traffic area (actually, many traffic jam happen) and vise versa. On the other hand, strict time synchronization makes some processing overhead.

Research on network simulation using virtual machines is concerned with "time synchronization". In discrete simulations, such as those dealing with human flows and traffic congestion, the simulation must proceed with all entities (people, cars, etc.) synchronized in time. If the simulation is carried out without synchronization, the credibility of the simulation results will be lost, for example, congestion does not occur in a place where people are supposed to be congested or in a place where cars are supposed to be congested. There

are two types of synchronization methods: one is the one that depends on the real time and the other is the one that is completely independent of the real time. References [10], [55], [26], [13] are real time dependent time synchronization methods. Reference [56] is a method of time synchronization using "virtual time" that is completely independent of real time.

In the case of my mobility simulation, strict time synchronization is not always needed because all simulators in my simulation environment obey the actual time, therefore, time skew between simulators is much smaller than it influences the reaction of the mobility. A modern operating system has a matured fair share scheduling algorithm. A Major Linux distribution (Ubuntu) use 250 Hz tick for preemption time. Preemption time means an interval time of context switching on a CPU. If there are more than two simulators on one CPU, There are time gap about approximately 4ms between these simulators in the moment of context switch. Considering the situation that a vehicle is running 100km/h and about to avoid an obstacle which is 5 meter ahead from the vehicle. During the 4ms gap, a vehicle will run about 10cm. I don't think that I should take care this time gap for strict mobility simulation, compared with introducing a complex strict time synchronization mechanism.

6.2 Connectivity to the Other Simulators

Simulation technology is used at various field and there are many kinds of simulation technology. According to the literature [20], The following list are the fields that simulation technology is used.

Table 6.1: Technology fields that uses simulation technology

Main fields	Sub fields
electrical and electronic	sound, material, nano-technology electromagnetic field analysis, VLSI design
machinery	material dynamics, material processing fluid dynamics, thermal engineering mechanical dynamics, measurement control production system, robotics, mechatronics computational dynamics, optimization space technology, traffic and logistics
environment and energy	area environment, disaster prevention energy (nuclear fusion, nuclear power, etc) urban planning
life science	life information (cell simulation, signal transmission) life cycle system, biomaterial, medical, welfare mechanics
human society	recognition, behavior, social system economics, finance, management, production risk, reliability, education
telecommunication network	network, wireless, protocol

I chose the several fields that are related to self-driving technology and describes the individual cases of cooperation in the following subsection.

6.2.1 Weak Coupling Analysis and Strong Coupling Analysis

Composite simulations have been used for many years for coupling simulations of machine parts and other applications, and there are two types of analysis methods, "strong-coupled analysis" and "weak-coupled analysis". Strongly-coupled analysis" is an analysis method with "interaction" in which the solution is obtained by using multiple analysis codes while comprehensively utilizing physical quantities, while "weakly-coupled analysis" refers to obtaining the physical quantities with one analysis code and then obtaining the solution based on the physical quantities with another analysis code. For example, it is used to

simulate the behavior of a car body by interlocking the behavior of the engine, transmission, and suspension of an automobile, and to verify the durability of a mechanism composed of different materials such as plastic and metal. In the following verification with specific examples, we will consider how our simulation environment and external simulations can be linked based on the concept of strong and weak coupling analysis.

6.2.2 Individual cases of Cooperation

Global Environment, Climate and Disaster Simulation

Climate and disaster simulations are closely related to automated driving simulations. The ad hoc network simulation we mentioned above is designed to verify whether an ad hoc network using communication equipment installed in a car can function properly as an alternative to the communication infrastructure when a large scale earthquake damages the communication infrastructure, such as base stations, and makes communication difficult. ones. Furthermore, in the event of a disaster, the self-driving function on the vehicle allows each self-driving vehicle to automatically stop the car at the side of the road or, if the ad hoc network cannot be properly configured as a whole due to the car being stuck and stopped, the appropriate ad hoc network can be configured so that the Simulations such as autonomous movement can also be carried out.

One of the leading climate and disaster simulations is a large-scale scientific simulation using a supercomputer. Reference [23] is a simulation of crustal deformation using the "K computer" supercomputer. This allows for more precise prediction of earthquake damage than ever before. Many simulations of the global environment, climate, and disasters, as exemplified in Reference (a), are often carried out by large-scale computers such as

supercomputers, and the simulations are carried out over a period of several days to several weeks, rather than being retrieved midway through the simulation process. Therefore, the results of the climate and disaster simulations are simply used unilaterally by the automated driving simulation, rather than cooperating with such simulations in real time to carry out a complex simulation.

The linkage with the climate and disaster simulation is a weakly coupled analysis. In this case, the climate and disaster simulation is the first to produce the results of the analysis, and the automatic operation simulation is carried out based on the results of the climate and disaster simulation. Specifically, in the case of cooperation with the earthquake damage simulation, the earthquake damage simulation is carried out first, and the information on road disruptions and the damage status of communication facilities are stored in the database of my simulation environment. Then, during the automatic operation simulation, an earthquake will be generated, which will disrupt the road and damage the communication facilities, and the automatic operation simulation under the disaster situation will be carried out. The function required here is the data conversion function to store the disaster simulation correctly in the database.

Recognition, Behavior and Social System Simulation

Simulation of cognition, behavior, and social systems is an important component of automated driving simulation. Cognitive-behavioral and social system simulations are used to examine how an individual's behavior affects the crowd and society as a whole, such as the relationship between rumor propagation and purchasing behavior or the effect of anxiety on evacuation behavior. able. When these cognitive, behavioral, and social simulations

are combined with my automated driving simulations, they can be used to recognize the behavior of pedestrians and have the automated car take evasive action, or to change the psychology of the passengers inside by the behavior of the automated car.

Typical simulations of cognitive, behavioral, and social systems include pedestrian simulations. Reference [30] is a representative pedestrian simulation. Like the pedestrian simulation in Reference [30], many of the simulations of cognitive, behavioral, and social systems take the form of agent-type simulations, such as my simulation environment, and many of these simulations are able to write out the results of their execution during the course of the simulation. In addition, such agent-type simulations often have interfaces that accept input from outside the simulation in order to change the behavior of individual pedestrians during the simulation. Therefore, in cooperation with cognitive, behavioral, and social system simulations, individual simulations are run in parallel while the simulation results are used with each other.

The linkage with cognitive, behavioral, and social system simulations is a strongly coupled analysis. As mentioned earlier, the linkage between my simulation environment and simulations of cognitive, behavioral, and social systems involves the mutual use of simulation results while running individual simulations in parallel. In a concrete case, an automatic car may take evasive action to avoid a collision with an oncoming pedestrian or cyclist, for example. When simulating a situation where the pedestrian takes evasive action and the self-driving car swerves in a direction that does not cause a collision with the pedestrian, the pedestrian's evasive action is stored in the database in my simulation environment, and the self-driving car recognizes it and at the same time, the automatic Pedestrians need to be aware of the direction of movement of the driving vehicle. The func-

tionality required here would require a program such as Formatter to store the pedestrian's avoidance behavior in a database as well as to enter the position and speed information of the automated vehicle into the pedestrian simulation.

Sound, Microwave and Peripheral Circumstance related Simulation

The simulation of the surrounding natural environment, such as acoustics and radio waves, is also an important component of an automated driving simulation. In the automated driving simulation, information is exchanged with other self-driving cars in the vicinity in advance to avoid collisions with each other at intersections with poor visibility. Since wireless communication is used for the information exchange, the simulation must simulate the radio wave environment in the vicinity. This will enable us to confirm how fast and how much data can be sent and received in real life. In addition, a simulation of the acoustic effect is required for vehicles equipped with sonar for collision prevention.

Typical simulations of the surrounding natural environment, including acoustics and radio waves, are radio wave propagation simulations such as qomet [14]. In the simulation of radio wave propagation, the attenuation function of radio waves is calculated with the parameters of the distance between two points where communication is carried out and the presence or absence of a shield. In the simulation of radio propagation, the attenuation function and other numerical calculations are used as a basis for calculating the radio strength between two points, and the effect of increasing the number of points and moving them to investigate how the communication between them is affected. The points are often assumed to be moved by pedestrians or cars, which means that they are essentially assumed to be used in conjunction with other simulators in advance.

Coordination with simulations of the surrounding natural environment, such as acoustics and radio waves, is a strongly coupled analysis. As mentioned earlier, the simulation of radio wave propagation is essentially assumed to be used in conjunction with other simulators. qomet, in addition to calculating the attenuation function of radio waves, also accepts input data such as the movement of points and the presence or absence of shielding. Using these parameters, the system calculates the strength of the radio signal at each time of day at a given point. When collaborating with my simulation environment, it is expected to use the attenuation function of radio wave propagation as a library, rather than using the simulation results with each other, as opposed to collaborating with cognitive, behavioral and social system simulations. This is due to the fact that simulations of the surrounding natural environment, such as acoustics and radio waves, are composed of a collection of subsets, such as radio wave intensity calculations between two points.

6.3 Domain Specific Cloud Management

I used a context of an application positively and indicated the fact that the application context gives us much more effective resource management. "domain specific cloud management" would be an appropriate words for my research. "Domain specific" in the cloud management field is often used as making a special language focused on easy configuration for an application. Ansible, Chef and Puppet are the major cloud configuration tools that use domain specific language (DSL) for making a specification of a cloud. The purposes of the existing DSL is to reduce the time to make a cloud configuration but they do not step in the performance improvement. My research offers such performance improvement,

through using the application context and indicates a next step for cloud management research.

6.4 Verification and Validation of computer simulation models

It is the most important whether or not my simulation environment can correctly simulate a driving environment for self-driving system. In the simulation research field, Verification and Validation (V&V) of computer simulation models are discussed for a long time. Verification implies to check the property that an appropriate mathematical model is used in our target simulation. Validation implies to check the fact that the output from the target simulation has statistically significance, compared with the output from a real system. In other words, verification concerns "accuracy of method", and validation concerns "validity of output" [48]. In a case of numerical analysis simulation, statistical methods is used for V&V. For example, in the case of computational fluid dynamics simulation, the following sequences are the basic tasks from modeling to running a computer simulation.

1. gives a mathematical model that represents a major part of a target system. A differential equation is often given for a mathematical model.

2. transforms a mathematical model into a computational model (ex. Finite Element Method : FEM).

3. make programs from the computational model and executes them on a computer.

In the above sequences, verification indicates to check whether or not a differential equation is appropriate, and validation is to check the output of a computer simulation

has statistical significance. validation will be achieved the comparison of two outputs. One is the result from computer simulation and another is the result from Physical simulation (referred in the figure 1.1). Therefore, V&V is familiar to a numerical simulation and it is difficult to apply to non numerical simulation. For example, multi agent model simulation is used to observe a macro behavior of many agents and there are no mathematical model that represents a major part of the target system. In the case of non-numerical simulation, V&V will be achieved by a decision of a knowledgeable people.

My simulation environment just offer a base infrastructure to aggregate many pseudo devices (called formatter in this dissertation) and simulators, therefore, verification and validation should be done by the development phase of each simulator and formatter. My simulation environment is a place where many simulation related components are aggregated and run a driving simulation and each component should guarantee the validation of their own output (and also the verification should be checked during the development phase of the simulator). Refer to the figure 1.2, a radio wave simulator could have a criterion of their simulation correctness that would be acquired by the comparison two outputs; one is the simulation output. and another is the measurement result on a field trial. Physics simulation could also verify the correctness of the simulation result, compared with the actual physics field trial. The other simulators (traffic and pedestrian) could receive certification that a knowledgeable people agrees that its simulation has a significant result. The only thing that I have to guarantee for correct simulation is the "reasonable delay" between simulation related components. In the chapter 3, I figured out that my simulation environment keeps the processing delay between the simulation components within a certain criterion that the user of my simulation environment can run a driving simulation without a bad influence.

Instead of whole verification and validation of my simulation environment, I can prepare a "visualization tool" for what property the simulation environment has. I describe a sample output of simulation property in the figure 6.1.

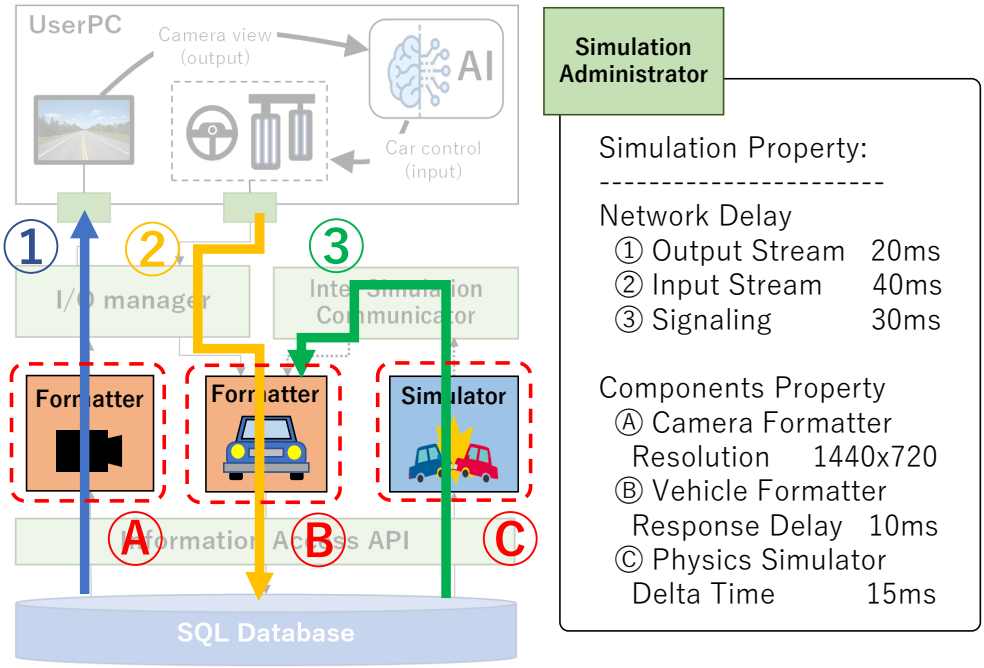

Figure 6.1: Example output of simulation property

My simulation environment has a simulation administrator component that coordinate a simulation, start the formatters and simulators, connect them with Inter Simulation Communicator, and show the simulation environment to User PC. Simulation Administrator can offer variety of property that a simulation environment has if these property is stored on a database with the simulation components. Users of the simulation environment can decide whether or not this environment is appropriate for their purpose, and also can change the components whatever they may want for their purpose.

Chapter 7

Conclusion

7.1 The Outcome of the Research

In the chapter 3, I suggested a distributed simulation environment that simulator and formatter is connected via a middleware and all data is stored in one database and shared to all simulation components. my simulation environment enable us to develop a simulator independently and swap it to another one in accordance with a test case. I also give a preliminary evaluation to my simulation environment and it reveals the fact that my simulation environment is feasible for self-driving simulation.

In the chapter 4, I suggested a deploy mechanism of VMs which was based of Voronoi decomposition of a simulated map. My mechanism executes several virtual machines on a same physical machine, which are “geographically” close to each other. My approach reduces the number of network sessions by one-sixth in the case of 500 VMs, compared with complete packing and round robin dispatching.

In the chapter 5, I suggested a novel approach which enables to assign priority information in accordance with the interaction information during the self-driving simulation. My approach can reduce the processing time of the jobs those are created by the self-driving AIs to one ninetieth. This result indicates the fact that my approach contribute the real-time processing during CPU overload.

7.2 Social Significance

As I mentioned it before in the section 1.1.3, there are many traffic accidents during driving a test of self-driving AI on public roads. These traffic accidents were caused by some unexpected situations when they test it on a computer simulation. One would be a fault of object recognition. Another is an unexpected pattern of interaction against vehicles or pedestrians. For a fault of object recognition, my simulation environment can offer a variety of object patterns because my environment can change a part of the simulators or formatters. For example, if a self-driving AI has possibility to make traffic accident caused by a fault of processing an image from camera, my simulation environment can change a variety of camera formatters which have different features (different resolution, put block noise on an image, etc). For an unexpected patter of interaction, my simulation environment can offer a variety of interaction pattern that can be made by a massive simulation. Thus, I am convinced that my simulation environment can reduce the self-driving related traffic accidents and promote the next motorized society.

7.3 A Prospect of the Massive and Complex Simulation Research Field

I suggested the three basic mechanism to give the simulation environment scalability. My approach is based on the operation of the plain information: map dividing and interaction prediction on roads. There should be many researches related to high performance computing. For example, hardware acceleration of SQL access, hot standby algorithm for creating a virtual machine, pass through mechanism of I/O between VMs and so on. These problems should be solved when my approach is implemented on an actual cloud. One of the largest problem is how I should apply my approach to three-dimensional space. My approach basically use the plain information. Voronoi decomposition is based on 2D map, and also prediction based priority setting is based on the traffic simulation on a 2D map. The calculation order of my approach is at most O(2), however, my approach must have more than O(3) calculation order if it deals with 3D information. Drone would be one of the typical application that my approach must deal with 3D information. At that time, I must have some researches how to reduce the calculation order.

Acknowledgement

I am grateful to all faculties who teach me with great patients. The supervisor professor Yoichi Shinoda. I learned many things from him, attitude for research, to play with technologies, and to enjoy coding. I am grateful to the sub advisor professor Yasuo Tan, the advisor of sub-theme, associate professor Kokoro Ikeda. I am also grateful to associate professor Ken-ichi Chinen he gave me many great advice, associate professor Fumio Machida he reviewed my PhD research. I am very grateful to associate professor Takahiro Koita, he reviewed my PhD research and also urged me to enter Japan Advanced Institute of Science and Technology. If I did not meet him, I would not be here and try to acquire PhD. I deeply appreciate for his support.

I am grateful to all classmates in the Tokyo satellite laboratory. They kindly discussed my research topics many times and often gave me a wonderful motivation. The classmates includes the following individuals, Hiroshi Abe, Eiichi Muramoto, Naomi Okumura, Yuki Unno, Takuya Kanamaru, Koki Iwata, Daiki Koezuka, Toru Makabe, Ryo Tsuruta, Hiroshi Takechi, Kentaro Kuribayashi, Kenichi Hanake.

I am grateful to my bosses and all colleagues. My boss, Toshiyuki Kanoh gave me a wonderful opportunities to go abroad to study. I realized the fact that I should acquire

PhD during the days in US. My boss, Tomoyoshi Sugawara supported my research for a long time and gave me many knowledge and advice. Thanks to my colleagues. They discussed my research topics many times and give me wonderful many suggestions.

It is my great pleasure that I am supported by my wonderful families. Father, Mother, sister, they always encouraged me to work and released my pressure when I met them. Father-in-law would be another supervisor of my research. He experienced the same situation and he knew what I was and what I thought. His advice encouraged me to work hard. my sons, you always gave me relax place to go back. I apologize both of you not to play with. You always cheerfully encouraged me to work with great patient.

In the end, to Sayaka, I am fortunate in having such a wonderful wife. I really appreciate your entire support, patient and love.

Presented Papers

Journal （reviewed）

- Akihito Kohiga, Yoichi Shinoda, Takahiro Iwai, Kozo Satoda, Creating Massive and Complex Ad Hoc Network Simulation Environment Based on Virtual Machine Technique, IPSJ Journal, 59(10), 1814-1826, 2018

International conference（reviewed）

- Akihito Kohiga, Yoichi Shinoda, A Deploy Mechanism for Virtual Machine Based Vehicular Ad Hoc Network Simulation, SpringSim '20: Proceedings of the 2020 Spring Simulation Conference, 1-12, 2020

Domestic workshop

- Akihito Kohiga, Yoichi Shinoda, Tomoyoshi Sugawara, Analysis of Response Time Fluctuation Factors on Network Function Virtualization , IPSJ SIG Notes 2014(9) 1 - 8, 2014

- Akihito Kohiga, Hiroya Kaneko, Nobuhiko Ito, Takanori Iwai; Application response time regulation system for auto mobility system, IPSJ SIG Notes 2017(4) 1 - 13, 2017

References

[1] Gstreamer. https://gstreamer.freedesktop.org/.

[2] Intelligent transport systems— Geographic Data Files (GDF)— GDF5.1—. https://www.iso.org/obp/ui/iso:std:iso:20524:-1:dis:ed-1:v1:en.

[3] List of self-driving car fatalities. https://en.wikipedia.org/wiki/List_of_self-driving_car_fatalities.

[4] Open Street Map. https://www.openstreetmap.org.

[5] Udacity's Self-Driving Car Simulator. https://github.com/udacity/self-driving-car-sim.

[6] Unity. https://unity.com/.

[7] C. E. Aguero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. L. Rivero, J. Manzo, E. Krotkov, and G. Pratt. Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response. *Automation Science and Engineering, IEEE Transactions on*, 12(2):494–506, 2015.

[8] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim. Core: A real-time network emulator. In *MILCOM 2008 - 2008 IEEE Military Communications Conference*, pages 1–7, Nov 2008.

[9] H. Alt, D. Hsu, and J. Snoeyink. Computing the largest inscribed isothetic rectangle. Technical report, Vancouver, BC, Canada, Canada, 1994.

[10] G. Apostolopoulos and C. Hassapis. V-eM: A Cluster of Virtual Machines for Robust, Detailed, and High-Performance Network Emulation. *14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pages 117–126, 2006.

[11] V. Autefage and D. Magoni. Nemu: A distributed testbed for the virtualization of dynamic, fixed and mobile networks. *Computer Communications*, 80:33 – 44, 2016.

[12] P. Bakkum and K. Skadron. Accelerating sql database operations on a gpu with cuda. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU-3, pages 94–103, New York, NY, USA, 2010. ACM.

[13] C. Bergstrom, S. Varadarajan, and G. Back. The distributed open network emulator: Using relativistic time for distributed scalable simulation. *Proceedings - Workshop on Principles of Advanced and Distributed Simulation, PADS*, 2006:19–25, 2006.

[14] R. Beuran, L. T. Nguyen, K. T. Latt, J. Nakata, and Y. Shinoda. Qomet: A versatile wlan emulator. In *21st International Conference on Advanced Information Networking and Applications (AINA '07)*, pages 348–353, May 2007.

[15] G. R. de Campos, P. Falcone, and J. Sjöberg. Autonomous cooperative driving: A velocity-based negotiation approach for intersection crossing. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1456–1461, Oct 2013.

[16] C. Dennl, D. Ziener, and J. Teich. Acceleration of sql restrictions and aggregations through fpga-based dynamic partial reconfiguration. In *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 25–28, April 2013.

[17] W. Dong. A time management optimization framework for large-scale distributed hardware-in-the-loop simulation. In *SIGSIM-PADS*, 2013.

[18] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[19] M. Fellendorf. VISSIM: A microscopic Simulation Tool to Evaluate Actuated Signal Control including Bus Priority. *64th Institute of Transportation Engineers Annual Meeting*, (January):1–9, 1994.

[20] J. S. for Simulation Technology. *Simulation dictionary (Japanese)*. CORONA PUBLISHING CO.,LTD., 2012.

[21] R. Fujimoto. Parallel and distributed simulation. In *2015 Winter Simulation Conference (WSC)*, pages 45–59, Dec 2015.

[22] R. M. Fujimoto. Parallel simulation: Parallel and distributed simulation systems. In *Proceedings of the 33nd Conference on Winter Simulation*, WSC ' 01, page 147–157, USA, 2001. IEEE Computer Society.

[23] K. Fujita, T. Ichimura, K. Koyama, H. Inoue, M. Hori, and M. Lalith. Fast and scalable low-order implicit unstructured finite-element solver for earth's crust deformation problem. pages 1–10, 06 2017.

[24] X. Gong. NavInfo HD maps make automated driving safer and more comfortable. https://www.itu.int/en/ITU-T/Workshops-and-Seminars/20180709/Documents/S3-P6-HD.pdf, 2018.

[25] G. Görbil and E. Gelenbe. Design of a mobile agent-based adaptive communication middleware for federations of critical infrastructure simulations. In *CRITIS*, 2009.

[26] A. Grau, S. Maier, K. Herrmann, and K. Rothermel. Time jails: A hybrid approach to scalable network emulation. *Proceedings - Workshop on Principles of Advanced and Distributed Simulation, PADS*, (June):7–14, 2008.

[27] A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[28] M. Hafner, C. D, C. L, and V. V. D. Automated Vehicle-to-Vehicle Collision Avoidance at Intersections. In *18th World Congress on Intelligent Transport Systems*, oct 2011.

[29] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 253–264, New York, NY, USA, 2012. ACM.

[30] D. Helbing, I. Farkas, P. Molnar, and T. Vicsek. *Simulation of pedestrian crowds in normal and evacuation situations*, volume 21, pages 21–58. 01 2002.

[31] HERE. HD Localization Model Road Model HD Lane Model HERE HD Live Map The most intelligent sensor for autonomous driving. Technical report, 2017.

[32] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Auton. Robots*, 34(3):189–206, Apr. 2013.

[33] R. Isermann, J. Schaffnit, and S. Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, 7(5):643 – 653, 1999.

[34] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner. Sumo (simulation of urban mobility) - an open-source traffic simulation. In *4th Middle East Symposium on Simulation and Modelling*, pages 183–187, 2002.

[35] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. Mason: A multiagent simulation environment. *SIMULATION*, 81(7):517–527, 2005.

[36] K. Massow, B. Kwella, N. Pfeifer, F. Häusler, J. Pontow, I. Radusch, J. Hipp, F. Dölitzscher, and M. Haueis. Deriving hd maps for highly automated driving from vehicular probe data. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1745–1752, Nov 2016.

[37] T. Mathew, K. C. Sekaran, and J. Jose. Study and analysis of various task scheduling algorithms in the cloud computing environment. *Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2014*, pages 658–664, 2014.

[38] T. McLean, R. M. Fujimoto, and J. B. Fitzgibbons. Middleware for real-time distributed simulations. *Concurrency - Practice and Experience*, 16:1483–1501, 2004.

[39] N. Minar, R. Burkhart, C. Langton, and et al. The swarm simulation system: A toolkit for building multi-agent simulations. Technical report, 1996.

[40] T. Osogami, T. Imamichi, H. Mizuta, T. Morimura, R. Raymond, T. Suzumura, R. Takahashi, T. Idé, T. Osogami, R. Raymond, and R. Takahashi. IBM Mega Traffic Simulator. Technical report, 2013.

[41] L. Pantel and L. C. Wolf. On the impact of delay on real-time multiplayer games. *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 23–29, 2002.

[42] C. Petschnigg, G. Breitenhuber, B. Breiling, B. Dieber, and M. Brandstötter. Online simulation for flexible robotic manufacturing. In *2018 7th International Conference on Industrial Technology and Management (ICITM)*, pages 88–92, March 2018.

[43] M. Pizzonia and M. Rimondini. Netkit: Easy emulation of complex networks on inexpensive hardware. In *Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities*, TridentCom '08, pages 7:1–7:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[44] PTV Planung Transport Verkehr AG. PTV VISSIM. https://www.ptvgroup.com/en/solutions/products/ptv-vissim/, 1992.

[45] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, number Figure 1, pages 679–686, 2009.

[46] D. Raychaudhuri, M. Ott, and I. Secker. Orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pages 308–309, Feb 2005.

[47] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[48] R. G. Sargent. Advanced Tutorials: Verification and Validation of Simulation Models. *Proceedings of the 2011 Winter Simulation Conference*, pages 183–198, 2011.

[49] T. C. Schelling. Dynamic models of segregation. *The Journal of Mathematical Sociology*, 1(2):143–186, 1971.

[50] H. Shimada, A. Yamaguchi, H. Takada, and K. Sato. Implementation and evaluation of local dynamic map in safety driving systems. *Journal of Transportation Technologies*, 05:102–112, 01 2015.

[51] N. Shiratori, F. Satoh, M. Saito, S. Ishihara, and T. Watanabe. *Simulation (Japanese)*. Kyoritsu Shuppan Co., Ltd., 2013.

[52] M. Treiber and A. Kesting. An open-source microscopic traffic simulator. *IEEE Intelligent Transportation Systems Magazine*, 2(3):6–13, Fall 2010.

[53] K. Uno and K. Kashiyama. Development of simulation system for the disaster evacuation based on multi-agent model using gis. *Tsinghua Science and Technology*, 13(S1):348–353, Oct 2008.

[54] V. Perrier. Cloonix. http://clownix.net/, 2007.

[55] E. Weing, F. Schmidt, H. Lehn, T. Heer, and K. Wehrle. SliceTime : A platform for scalable and accurate network emulation. *Communication*, page 19, 2011.

[56] S. Yoginath, K. Perumalla, and B. Henz. Virtual machine-based simulation platform for mobile ad-hoc network-based cyber infrastructure. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 12:439–456, 10 2015.

[57] Yuni Xia and S. Prabhakar. Q+rtree: efficient indexing for moving object databases. In *Eighth International Conference on Database Systems for Advanced Applications, 2003. (DASFAA 2003). Proceedings.*, pages 175–182, March 2003.

[58] M. Zec and M. Mikuc. Operating system support for integrated network emulation in imunes. *1st Workshop on Operating System and Architectural Support for the on Demand IT InfraStructure (OASIS)*, pages 3–12, 01 2004.

[59] T. Zhang, S. Zhao, B. Cheng, B. Ren, and J. Chen. Fep: High fidelity experiment platform for mobile networks. *IEEE Access*, 6:3858–3871, 2018.