CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Large-Scale Traffic Simulation Through Distributed Computing of Paramics

Henry X. Liu, Wenteng Ma,
R. Jayakrishnan, Will Recker

California PATH Research Report
UCB-ITS-PRR-2004-42

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California. This report does not constitute a standard, specification, or regulation.

Final Report for Task Order 4142

October 2004

ISSN 1055-1425

CALIFORNIA PARTNERS FOR ADVANCED TRANSIT AND HIGHWAYS

Large Scale Traffic Simulation Through Distributed Computing of Paramics

Henry X. Liu
Wenteng Ma

Department of Civil and Environmental Engineering
Utah State University

R. Jayakrishnan
Will Recker

Institute of Transportation Studies
University of California, Irvine

California PATH Program
Final Research Report for TO4142

ABSTRACT

Simulation modeling is an increasingly popular and effective tool for analyzing transportation problems, which are not amendable to study by other means. We examine the need for parallel or distributed simulation approaches from the need for computational speed-ups, availability of options towards that, and then at the need to distribute the effort to develop network simulation contexts and datasets. After an overview of the general techniques for the distributed discrete-event simulation and previous efforts on the distributed traffic simulation, we present the general architecture of the proposed distributed modeling framework. Two categories of modeling strategies, namely, light global control / independent subnets vs. heavy global control / coordinated subnets are described. We have implemented the distributed scheme of light global control / independent subnets and the implemented details, such as communication techniques and vehicle transferring across the boundary of two subnets are discussed. Unlike the previous studies using the dedicated high performance machines, our efforts are to utilize the low-cost networked PCs that are commonly available. By using the API supported by off-the-shelf Paramics software, we are able to distribute the computational load of microscopic simulation to multiple single-processor PCs without access the proprietary source codes of the simulation program. Performance testing and analysis of the implemented prototype demonstrate that the proposed framework is very promising.

Keywords: Microscopic Traffic Simulation, Paramics, Distributed

TABLE OF CONTENTS

ABSTRACT ..... **i**
TABLE OF CONTENTS ..... **ii**
LIST OF TABLES ..... **iii**
LIST OF FIGURES ..... **iii**
EXECUTIVE SUMMARY ..... **iv**
INTRODUCTION **1**
- **The need for faster simulations:** 1
- **Would future hardware improvements be enough for fast simulations?** 2
- **Need for distributed sub-network schemes for simulation management:** 2

LITERATURE REVIEW **3**
- **Overview of Distributed Discrete-Event Simulation** 3
- **Previous Efforts on Distributed Traffic Simulation** 4
- **Paramics Simulation Tools** 5

DISTRIBUTED MODELING FRAMEWORK **6**
- **General Architecture** 6
- **Light Global Control / Independent Subnets** 7
- **Heavy Global Control / Coordinated Subnets** 8
- **Communication Technologies** 9
- **Load Balancing** 9
- **Synchronization Mechanism Using Conservative Time Window (CTW)** 10

PERFORMANCE TEST AND ANALYSIS **11**
- **Prototype for Light Global Control / Independent Subnets** 11
- **Example Problems and Test Environment** 11
- **Computation Load Study** 12
- **Benefits from the Distributed Modeling Framework** 12
- **Consistency between Distributed and Sequential Simulations** 13

CONCLUDING REMARKS **13**
REFERENCES **14**

LIST OF TABLES

1. Computation Load percentage of those other than Updating Vehicle Information
2. Computation Load Contribution of Updating Vehicle Information
3. Speedup of the Example Network 1
4. Arrival number of Vehicles Comparing with the Original Demand with a 2-Server for Grid Network.

LIST OF FIGURES

1. The space-time graph
2. General Distributed Modeling Architecture.
3. Prototype of the Distributed Modeling Framework.
4. Transfer Vehicle Forecasting.
5. (a) Example network 1 — Double Logan Network.
   (b) Example network 2 — Large-scale grid network.
6. Speedup of the Example Network 2 with Scenario 1.
7. Speedup comparing with Scenario 1 and Scenario 2 of the Example Network 2.

EXECUTIVE SUMMARY

Although parallel computing capability of Paramics is critical to large scale traffic simulation, Paramics Parallel is not commercially available at present. The goal of this project is to develop a software system, using WinSocket communication protocol, to allow Paramics user to distribute computation load demanded by large-scale microscopic traffic simulation over a number of homogeneous processors. The parallel environment for Paramics could be either a multi-processor machine, with shared memory or networked workstations connected by Ethernet. The first one usually has less communication overhead than the second since the data transmitting speed of computer BUS is faster than that of Ethernet. In this project, we use the low-cost networked PCs because they are commonly available.

Distributed computing systems predominantly use client-server model. Under this model, one processor called the client coordinates the other processors in the system, called servers, to function as a single computational unit. Coordination is performed through the message passing among the client and the servers. Therefore, the general distributed architecture will include at least a controller (client) and several sub-network simulators (severs). Although the controller may have various tasks related to coordinating the traffic simulation itself, the essential task from a computational architecture standpoint is the synchronization of the time in each sub-network, either at every simulation time-step or at specified intervals of times. Depending on different simulation strategies, the controller-simulator architecture could lead to different styles of design, including light global control / independent subnets and heavy global control / coordinated subnets, as described in the following.

The light global control / independent subnets design is the simple form of distributed simulation. In this case, each sub-network simulator has its own origin-destination demand matrix and its own route tables, but its simulation clock time is synchronized by the controller. In other words, if the time synchronization from the controller is removed, each sub-network simulator should be able to perform its simulation independently. The only task for the controller is to synchronize the time clock for each sub-network. This design features easy implementation since the vehicle routing has been taken care by the individual simulator. On the other hand, since each simulator knows only the local traffic condition in the sub-network, without knowing the big picture of the large network, this design may suffer from the unpractical or unrealistic routes taken by some vehicles. Therefore the congestion pattern from the simulation may be distorted from reality.

The design with a heavy global controller and coordinated subnets is at the other end of the spectrum, compared to the previous design. In this case, not only will the controller synchronize the time clock of simulators but also contain the global abstract network, global O-D matrix and global routing table. The controller will control the vehicle generation in the sub-networks and the individual vehicles' paths. The local traffic condition in the sub-networks will be reported back to the controller and used in the dynamic update of the global routing table. When a vehicle comes to the boundary of the

originating sub-network, the controller will notify the receiving network to generate an identical vehicle, and continue to route the vehicle to the destination. Here, each sub-network simulator is only used to update the individual vehicle's location according to the car-following, lane-changing and gap-acceptance models, which are the most time-consuming parts in microscopic simulation.

In this project, we implement the prototype of the distributed modeling framework for the light global control / Independent Subnets. Two example networks were built to test the performance of the proposed distributed modeling framework. Example network 1 was coded based on the real network, but expand artificially for the computational load study. The origin real network is part of the Main Street of Logan, UT, including one main arterial, 10 signalized intersections and 22 O-D demand zones. We extended the network by double the links and O-D demand zones. Example Network 2 was artificially designed as a large-scale grid network. The original network is coded with 22 arterials, 120 signalized intersections, 44 O-D demand zones, and the peak hour demand of the whole network is 33840 vehicles. Two scenarios were tested with this grid network. The grid network was evenly divided in 2 parts for Scenario 1 and 3 parts for Scenario 2. Testing was carried on three desktops, the double Logan network and Scenario 1 of the grid network were carried at two desktops each with one 2.8 GHz CPU processor and 1 GB RAM. Scenario 2 of the grid network was carried with one additional workstation that has 3.2 GHz CPU processor and 2 GB RAM. All of the operation systems are Windows XP Professional. Each client computer was carried one part of the origin network in Paramics Modeler Version 4.2.1. All other traffic conditions and simulation configurations at each simulator are the same.

The speedup results from these testing scenarios are very promising. For the Example Network 1, as the synchronized time window increase, the speedup will increase from 0.73 up to 1.52 with time-step 2 per second and from 1.23 up to 1.65 with time-step 8 per second. For the Example Network 2 with Scenario 1, the benefit from the proposed distributed modeling framework in this scenario was up to 2.4 with 2 per second time-step, and up to 2.7 with 8 per second time-step with the 1 minute dynamic feedback interval. The reason why the speedup factor is higher than 2 is that the computational time for the dynamic feedback update is sub-linear, i.e., the calculation time for the whole network for the dynamic feedback may be higher than 2 times comparing that of a subnetwork with half size. For Scenario 2, we will get more benefits with more distributed servers. However, due to the communication load will be significant increase as the number of clients increase, some benefits from the vehicle update will be counteracted. Which means the benefit form increasing the number of distributed client may be limited.

The research demonstrated in this paper is a light global controller / impendent subnets, which is only the first part of this proposed distributed modeling framework. To achieve more accurate distributed simulation results, the heavy global controller / coordinated subnets design should be considered in the further research, including develop the algorithm for global routing, network decomposition, and etc.

INTRODUCTION

Simulation modeling is an increasingly popular and effective tool for analyzing transportation problems, which are not amendable to study by other means. In the transportation simulation field, there is some general agreement that micro-simulation, i.e., a computational resolution down to the level of individual travelers, is now a viable alternative and may be the only answer to a wide variety of problems. A traffic simulator for dynamic traffic management plays two distinct roles: as an off-line evaluation / design tool and as an on-line control / guidance tool. Both roles could be computationally intensive and demand fast simulator to fulfill their tasks. The need for parallel or distributed simulation approaches can be understood from examining the need for computational speed-ups, availability of options towards that, and then at the need to distribute the effort to develop network simulation contexts and data sets, as below.

The need for faster simulations:

Three primary types of simulation applications immediately show the need for faster simulations:

(1) Off-line Planning: Many types of localized traffic jams, which can only be produced on the micro-simulation level, affect people's modal and route choice. Furthermore, the environmental impacts of highway design alternatives are almost entirely dependent on the details of such congestion phenomena. The traditional planning models based on simple link travel time functions are proven to show unacceptably erroneous emission impact and many planning studies have ended up even in court litigation on this count. Integrated planning using microscopic modeling already requires efficient simulations of large networks of the kind needed for planning exercises. The simulation may have to loop several times back and forth between a route planning and a micro-simulation, thus producing a need to execute the micro-simulation several times instead of only once until a result is achieved.

(2) Online Control/Guidance and Emergency modeling: The online simulation approach is motivating a new realm of traffic control opportunities to influence the dynamic traffic conditions in congested networks. However, online simulation applications such as prediction for traffic control require the model to run much faster than real time. Existing sequential simulation processing in a single processor computation environment does not meet the intensive computational requirement for online data processing, especially in the case of modeling for real-time operations such as rerouting around accident sites. This is even more important if modeling is to be used for emergency management requiring area-wide evacuations.

(3) Monte Carlo Analysis: The apparent global stochasticity of traffic is captured in most modern simulation approaches. In order to generate credible results, multiple simulation runs are a must. There is a potential danger of analysts not repeating the simulations enough times, and using spurious results in making potentially costly decisions. Once again, the speed of simulations becomes critical.

Would future hardware improvements be enough for fast simulations?

Although Moore's law (the speed of a microprocessor would double approximately every 18 months) has been proven remarkably accurate to date *(1)*, with the complexity of the systems we are interested in simulating also keep increasing. Given the increasing network size requirements for analysis, it would remain impossible in the near future for a single processor to provide satisfactory simulation performance. For comprehensive traffic solutions for route-level and area-wide congestion amelioration, the analysis network sizes are often several miles or tens of miles in length and breadth. That is, networks of even 20 or 30 times the size/area of current (local) analysis networks need to be simulated. Furthermore, for online analysis of multiple solutions, we need another order of magnitude speed-up in the operations. Even a Moore's law extrapolation would require more than a decade for such a 100-fold increase in computational capabilities on singles computers. This implies that harnessing additional processors in parallel and decomposing the problem domain into sub-domains is an option of promise, and perhaps the only solution.

Although dedicated high performance computing systems (for example, shared memory multiple-processor workstation) can significantly reduce the computational time, most transportation agencies cannot afford them. In addition, a given sequential simulation program may have to be rewritten to exploit more parallelism to leverage the multiple processors. This will certainly add extra costs for the software development and these costs are usually not justified because of the small market. The most optimistic and affordable computational environment in the near future for most transportation agencies is a network of personal computers (PCs) connected by local area network (LAN). By distributing the computational load demanded by large-scale simulation to the inexpensive networked PCs, our goal is to relieve the computational burden and speedup the simulation.

Need for distributed sub-network schemes for simulation management:

In addition to the above issues, decomposed and distributed simulation schemes are a requirement in incrementally developing network data sets and debugging the simulation cases in any operational computer environment. For instance, it is nearly impossible to develop a network data set for all of Los Angeles basin or the San Francisco bay area without significant decomposition of simulation efforts across analysts and modeling personnel. Ensuring the integrity of the data sets will also require techniques that would allow smaller sub-areas to be tested with larger areas that they are part of, with the flow-through traffic modeled properly. To effectively "stitch" together sub-network data sets and to test their integrity distributed simulations are required. In other words, the capabilities developed for faster simulations will also yield capabilities for managing the simulation data set environment better. No effective framework with such comprehensive approaches to modeling practical networks exists in the traffic analysis community as yet. This is an added practical reason to develop abilities to set up simulation frameworks across robust distribution platforms.

In this paper, we propose a distributed modeling framework for the large-scale microscopic traffic simulation. In our proposed distributed simulation environment, the target large network will be divided into sub-networks, and each sub-network will be simulated on a separate desktop PC. Our distributed computing environment consists of a network of PCs operating under Windows XP and connected by a 100 Mbps LAN within a client –server framework. This type of computing environment is low-cost and commonly available to the transportation agencies, therefore the proposed framework can be adopted readily. We have selected Paramics (PARAllel MICroscopic Simulation) as the simulation platform and developed a software toolkit using Application Programming Interfaces (API) to demonstrate the distributed modeling framework.

This paper is organized as follows. We first offer a brief overview of the general techniques for the distributed discrete-event simulation. Previous efforts on the distributed traffic simulation are also summarized. We then present the general architecture of the proposed distributed modeling framework. Two categories of modeling strategies, namely, light global control / independent subnets vs. heavy global control / coordinated subnets are described. We have implemented the distributed scheme of light global control / independent subnets and the implemented details, such as communication techniques and vehicle transferring across the boundary of two subnets are discussed. Performance testing and analysis of the implemented prototype are followed. Finally, we offer concluding remarks and future research directions in the last section.

LITERATURE REVIEW

Overview of Distributed Discrete-Event Simulation

A simulation model can be viewed as a representation of the physical system under simulation. Discrete Event Simulations are characterized by discrete-state models (as opposed to continuous –state models) and the event-driven approach. The discrete event simulation can be further classified according to how the simulation time advances. In time-driven discrete simulation, the simulation time is incremented by a constant time step. In the event-driven approach, the increment of simulation time is triggered by the next earliest occurring event. If the simulation logic is executed in order of the events' simulated time of occurrence; i.e., the simulation is sequential. If the computational load is high, the sequential simulation could be slow. To circumvent this, techniques have been developed to "distribute" different parts of a computing task across individual processors at the same time, or in "parallel", and thus reduce the overall time to complete the task.

Distributed (or parallel) simulation is an application of distributed computing which aims at decreasing the computational time by engaging different processors of a multiprocessor system or different computers of a network to share the workload of a simulation program when latter is executed *(2)*. Seminal work of parallel discrete-event simulation dates back roughly 20 years. There are many conceivable ways of splitting up

a dynamic simulation to distribute its work over different processors, such as distributed functions, distributed events, and domain decomposition (for a comprehensive review, please refer to *11*). Among these classes of approaches, the domain decomposition shows the greatest potential and is considered the most promising approach to perform simulation in parallel.

Domain decomposition is based on the view that a simulation execution is equivalent to filling in a two-dimensional region, with one dimension representing the simulated time and another dimension the state variables, as shown in Figure 1(a). According to this view, the space-time domain can be decomposed along the time dimension, as illustrated in Figure 1(b), or along the space dimension, as illustrated in Figure 1(c). For space decomposition (or distributed simulation), the simulation model is decomposed into a number of sub-models or components in the space domain. Each component is assigned a process, where may be run on one processor. This decomposition is attractive because it is applicable to any model and shows the greatest potential in offering scalable performance in large models. For time decomposition (or time parallelism), the domain is partitioned into a number of intervals. Each process is assigned an interval and is responsible to compute the values of the state variables within that interval. With this approach, the simulation mechanism must ensure that the state of the system at the end of one interval match the system state at the beginning of next interval.

Microscopic traffic simulation is usually a time-driven discrete-event simulation, meaning that the state variables in the simulation, such as vehicle position and speed and traffic light status, are updated every constant time step. As we will discuss later, we adopt the space decomposition techniques to distribute the simulation across separate processors. The time decomposition is not readily applicable but deserves further research. We should note that the terms distributed simulation and parallel simulation are synonyms in this paper since we apply the space decomposition approach.

Previous Efforts on Distributed Traffic Simulation

There are a few traffic simulation applications have adopted distributed computing techniques, including Transportation Analysis and Simulation System (TRANSIMS) *(3)*, Advanced Interactive Microscopic Simulator for Urban and Non-Urban Networks (AIMSUN) *(4)*, and Parallel Microscopic Simulation (Paramics) *(5)*. The distributed algorithms are incorporated within the source codes of these applications and the computational workload can be shared among different processors or computers. In the following, we will briefly review the distributed simulation methodologies of TRANSIMS and AIMSUN, but focus on Paramics we use it our simulation platform in this paper.

TRANSIMS is an integrated system of travel forecasting models designed to give transportation planners information on traffic impacts, congestions, and pollution.

Distribution implementation in TRANSIMS is based on a domain decomposition principle, where the network is partitioned into domains of approximately equal size, with each CPU of the distributed computer responsible for one of these domains *(6)*. The microscopic simulation procedure uses a cellular automata (CA) technique for representing driving dynamics, and it is reported that using CA helps with the design of a distributed simulation update.

AIMSUN is a microscopic simulation program originally developed as a sequential, but later reported to distributed computers *(4)*. For distributed simulation implementation, AIMSUN uses a sequence of instructions that executed within the context of a process to handle a group of entities that need to be updated at every time step. A process can be executed in distributed by the multiple processor/computer system.

Paramics Simulation Tools

Paramics is a suite of microscopic simulation tools used to model the movement and behavior of individual vehicles on urban and highway road networks *(7)*. It simulates the components of traffic flow and congestion, and presents its graphical animation output simultaneously for traffic management and road network design. One important feature of Paramics is that it allows the user to customize many features of underlying simulation model through API. In general, the API functions can be used to override and extend the default functionalities or logic, and obtain and set the simulation parameters from Paramics. Through the API, relevant communication functions are programmed to interface with Paramics. In addition, Paramics is able to generate unique vehicle that all of its parameters, e.g. vehicle type, initial speed, releasing time, driver aggressive, and etc., are defined by the user, hence it is used to transfer identical vehicles from one sub-network to another sub-network in this research. Comparing with other traffic simulators, the new Paramics Version 4 has undergone a significant overhaul. The new V4 API is more structured, easier to use, and inherently safer. The breadth of V4 API has also been extended. So Paramics V4 is selected to test the proposed methodology in this paper.

Paramics was developed originally for a shared-memory Connection Machine CM-200 with 16,000 processors in 1992, using a data-distributed approach and being able to simulate approximate 200,000 vehicles on 20,000 miles of road lanes *(5)*. To make good use of CM-200, the data must be in a parallel array form so that operations can occur in parallel on the elements of the array. The approach used to build a parallel data framework for the simulation process was to associate a parallel item of data with each link in the network. However, with more complex vehicle dynamics, the artificial constraints imposed by the parallel data structure for CM-200 become increasingly significant.

In 1995, based on the previous success, Paramics was further developed using message-passing interfaces (MPI) and was targeted on a 256-node CRAY T3D *(5)*. This solution, named as Paramics-MP, could model 120,000 vehicles at three times the real-time rates on 32 nodes of the T3D. Paramics-MP is made up of a number of components: the simulator is the computational working unit; the concentrator gathers information

from simulator elements and merges it into a single data stream to the visualizer; the visualizer is the visualization module and runs on an SGI graphic workstation; the control panel is an extended graphical user interface, communicating with the visualizer and concentrator.

Another notable work on the Paramics parallelization is from the group at the National University of Singapore *(2)*. The idea was to divide the network into several regions and simulate under different instances of the program simultaneously, allowing transfer of vehicles at the boundaries of different regions. The method was implemented in Paramics API on a multi-processor UNIX System. Testing was carried out on a Sun E3500 Server system with 4 processors (450 MHz each) and shared memory (3GB RAM). They reported that the results showed an increase in speed ranging from 1.50 to 2.25 times when using two processors and from 1.75 to 3.75 times when using three processors, compared with the speed of simulation without distributed execution.

It must be noted that the above parallel simulation efforts are quite different in nature with what is proposed here. The majorities of them are targeted on the dedicated high performance system with multi-processor and share memory running UNIX system. Cares must be taken for the programming of a parallel code to prevent simultaneous access of same data in the share memory. The realization of distributed simulation usually requires the access the proprietary source codes of simulation software. Instead, our efforts are to utilize the low-cost networked PCs that are commonly available. By using the API supported by off-the-shelf Paramics software, we are able to distribute the computational load of microscopic simulation to multiple single-processor PCs without access the proprietary source codes of the simulation program. The methodology proposed in this paper is a distributed modeling framework especially suitable for path-level computations across sub-network simulations.

DISTRIBUTED MODELING FRAMEWORK

General Architecture

Before deciding the distributed architecture, CPU time consumed at each stage of the simulation process needs to be measured in order to identify which of them should be distributed. From the experience of parallelization efforts other traffic simulation software and also verified by the computation load study later in this paper, individual vehicle updating at each time step is most time consuming, and traffic detection and control is the second. Therefore, the basic distributed architecture is to decompose the large network into several sub-networks, and each sub-network will be simulated on a separate desktop PC.

Distributed computing systems predominantly use client-server model. Under this model, one processor called the client coordinates the other processors in the system, called servers, to function as a single computational unit. Coordination is performed through the message passing among the client and the servers. Therefore, the general distributed architecture will include at least a controller (client) and several sub-network

simulators (severs). Although the controller may have various tasks related to coordinating the traffic simulation itself, the essential task from a computational architecture standpoint is the synchronization of the time in each sub-network, either at every simulation time-step or at specified intervals of times. To synchronize the simulation time, the controller will have to be able to start and stop the sub-network simulation at any time. As an example of the type of traffic simulation issues to be handled by the "master network" simulation in the controller, consider a large network divided into several sub-networks, where there are some boundary zones existing in two or more sub-networks. A destination zone in one sub-network may correspond to an origin zone in another sub-network. Therefore, look-up tables such as one containing the information of boundary zones and their corresponding ownerships should be established in the controller computer.

During a simulation run, the controller and simulators communicate over the distributed platform. The sub-network simulators act as slaves to the controller. During a time step of simulation or certain time interval, a simulator executes a non-blocking loop (asynchronous communication) while waiting for a new request from the controller. A request is simply a message associated with a specific task. When the request arrives into a sub-network simulator, it starts with an execution of the corresponding sequential code. When the request task is completed, a notification is sent back to the controller. When all simulators are “checked in”, the simulation master clock advances by one step and broadcasts the new times to every simulator in the system. Each simulator then proceeds until it reaches the master clock time. A pictorial description of the scheme to be used for distributed processing is shown in Figure 2. Depending on different simulation strategies, the controller-simulator architecture could lead to different styles of design, including light global control / independent subnets and heavy global control / coordinated subnets, as described in the following.

Light Global Control / Independent Subnets

The light global control / independent subnets design is the simple form of distributed simulation. In this case, each sub-network simulator has its own origin-destination demand matrix and its own route tables, but its simulation clock time is synchronized by the controller. In other words, if the time synchronization from the controller is removed, each sub-network simulator should be able to perform its simulation independently. The only task for the controller is to synchronize the time clock for each sub-network.

Communication between subnet simulators is required when an object (a vehicle) moves from one sub-network to another. A message is sent from the originating simulator to the receiving simulator describing the object, and the “ownership” of the objects is thus transferred. Once the transfer is confirmed, the vehicle object disappears at the destination zone of the originating sub-network, and the corresponding vehicle will be generated from the origin zone in the receiving sub-network. The destination of the transferred vehicle is set randomly in the receiving sub-network, but the probability of selecting a particular destination zone is equal to the ratio of origin-destination demand

with total demand generated from the origin zone. Once the destination of the transferred vehicle is selected, the transferred vehicle will be taken as one of the vehicles generated from this particular O-D demand.

This design features easy implementation since the vehicle routing has been taken care by the individual simulator. On the other hand, since each simulator knows only the local traffic condition in the sub-network, without knowing the big picture of the large network, this design may suffer from the unpractical or unrealistic routes taken by some vehicles. Therefore the congestion pattern from the simulation may be distorted from reality.

Heavy Global Control / Coordinated Subnets

The design with a heavy global controller and coordinated subnets is at the other end of the spectrum, compared to the previous design. In this case, not only will the controller synchronize the time clock of simulators but also contain the global abstract network, global O-D matrix and global routing table. The global abstract network is a simplified network from the original large network, and used only for routing purposes. The controller will control the vehicle generation in the sub-networks and the individual vehicles' paths. The local traffic condition in the sub-networks will be reported back to the controller and used in the dynamic update of the global routing table. When a vehicle comes to the boundary of the originating sub-network, the controller will notify the receiving network to generate an identical vehicle, and continue to route the vehicle to the destination. Here, each sub-network simulator is only used to update the individual vehicle's location according to the car-following, lane-changing and gap-acceptance models, which are the most time-consuming parts in microscopic simulation.

The benefit from this design is that vehicle's origin-destination and its path are all controlled at the global level, as opposed to the local level in the previous design. In this aspect, the design is similar to the simulation over single processor in term of routing, with the distinction of updating vehicle's location over distributed processors. Although the communication load between the controller and simulators is also significant higher than that of previous design, which may slow down the simulation.

The heuristic routing approach could be taken to reduce the communication overhead. In this case, the controller will still have the global abstract network, global O-D matrix and global routing table. Each simulator also has its own local routing table. When a vehicle is generated in the sub-network, if its origin and destination belong to different sub-network, its temporary destination in the originating sub-network will be determined from the global routing table, but its path in the originating sub-network will be determined locally from the local routing table. Therefore, instead of routing every individual vehicle at the global level, the revised design allows a vehicle's route calculated at the local level. Significant communication overhead will be reduced in this case. However, this heuristic approach doesn't solve the routing problem in the first approach completely in that the vehicle's temporary destination determined from the global routing table may be changed in the dynamic assignment case.

Communication Technologies

There are several communication technologies that are popular used for distributed computation, including Distributed Component Object Model (DCOM), Common Object Request Broker Architecture (CORBA), and Windows Socket Programming (Winsock).

DCOM *(8)* is developed by the Microsoft Cooperation based on the Windows platform environment. DCOM allows for peer-to-peer communications between computers, and it allows for greater flexibility in the Windows environment. DCOM allows developers to write to communication services rather than building connectivity directly into the applications each time. It serves the same purpose that an Object Request Broker (ORB) does in the rival distributed computing framework, the Object Management Group's CORBA standard.

The standard CORBA includes three levels, including ORB, public object services, and public applications. Generally, an ORB enables communication between clients and objects, transparently activating those objects that are not running when requests are delivered to them *(9)*. Based on ORB, CORBA defines several public object services, like naming service, trading service, event service, and etc. The public applications are the framework, which supports the services that be directly used by the users. CORBA is supported on almost every combination of hardware and operating system in existence.

Windows Sockets enables programmers to create advanced Internet, intranet, and other network-capable applications to transmit application data across the wire, independent of the network protocol being used. It defines a standard service provider interface (SPI) between the application programming interface (API), with its exported functions and the TCP/IP protocol stacks *(10)*. Winsock is a lower level but still effective communication technique comparing with DCOM and CORBA, which are also based on the TCP/IP protocol. Considering the complexity of the implementation of the DCOM and CORBA techniques, Winsock programming is employed in the proposed methodology.

Load Balancing

Since synchronous communication is used among simulators and controller, each simulator can only run as fast as the slowest one. So a proper and balanced decomposition of the network is critical to the overall performance. The computation load study later in this paper shows the total computational requirement for a microscopic traffic simulation is dominated by the number of vehicles in the network at any time, the ideal division of network is to create N regions that each has exactly V/N vehicles, where V is total number of vehicles in the simulation and N is the target number of processors. The speed-up performance of the distributed processing is also dependent on the communication to computation overhead: if there are a large number of communication operations for each computational operation, the overall process will reduce in speed .In

order to minimize the communication overhead, distributed simulations require methodological decomposition of the large network to find a subdivision where there are as few boundaries as possible and the computational load is spread evenly across the processors. One of our ongoing researches is to design a multi-objective optimization program to decompose the large network

Synchronization Mechanism Using Conservative Time Window (CTW)

Parallel simulation based on discrete-event model fundamentally requires synchronous processing at every simulation clock cycle because causality constraints have to be observed in order to ensure the correctness of the simulation. Violating causality constraint means that the future can affect the past. This can results in anomalous behavior and consequently incorrect simulation. It is the responsibility of the synchronization mechanism to ensure the proper and correct interaction among the sub-network simulators. Synchronization schemes of parallel simulation broadly fall into two categories – conservative and optimistic – according to the ways they adhere to the local causality constraint *(11)*. Conservative approaches avoid all possible causality errors by strictly adhering to the local causality constraint. Optimistic approaches, on the other hand, attempt to exploit the nonzero probability of producing no causality error by not strictly adhering to the local causality constraint. But the occurrences of causality errors may lead to the rollback simulation to restore the simulation to a correct state.

In the research, we adopt the conservative approach to ensure the causality constraint. However, in order to reduce the synchronization overhead, we specify a time window that each sub-network simulator is independent within these windows and can be processed concurrently. The synchronization work will be done at the end of each conservative time window, which is greater or equal to the constant simulation time step. The problem is how to choose a proper time window. If the synchronization time window is too short, such as the constant simulation time-step, then the faster simulators will always waiting the slowest one and the synchronization overhead will make the benefits from the distributed simulation less. If the synchronization time is too long, the local causality constraint may be significantly violated, meaning that when the vehicle transferred from the slower simulators to the faster one, the vehicle may transferred behind. The simulation would thus not be carried out correctly.

To circumvent this problem, we introduce the transfer vehicle forecasting technique. Figure 3 shows the vehicle-transfer status between sub-networks. Instead of detecting the information and transferring vehicle at the boundary of the sub-networks (Point B), a detector is put ahead of the boundary (Point A), collecting and transferring the vehicle with a couple of simulation time ahead, called forecasting time here, which depends on the distance of the detector to the boundary, the speed of the vehicle, and etc. Then we can transfer the vehicle correspond exact to the real world, nor ahead or behind, even in the case that the synchronization time clock is not small.

PERFORMANCE TEST AND ANALYSIS

Prototype for Light Global Control / Independent Subnets

Figure 2 shows the prototype of the distributed modeling framework for the light global control / Independent Subnets. First, the controller will start the communication client and open a socket to listen and send messages, and then all the sub-networks on the sever-computers will be loaded through remote control. Meanwhile, the sever-computers will automatic establish the connections with the client. After all the sub-networks are loaded and all the servers are successful connected with the controller client, Paramics will start the simulations simultaneously. During each simulation run, the client and the servers will communicate through the Windows Socket platform. There are two types of message will be transferred, one is the synchronized information, and the other is the vehicle transfer information. The server-computer usually has different simulation speed due to variety reasons, such as different sub-networks, different number of vehicles in the simulations, different processor configurations, and etc. In order to synchronize all the server simulators, the faster simulators need to wait the slower ones after each synchronized time window that pre-defined by the user, like 30 seconds. The synchronized information is collected by the controller and feedback to the subnet simulators using Paramics APIs via the Windows Socket platform. Similarly, once a vehicle arrive the boundary of the sub-networks, the vehicle information will be sent from the "upstream" simulator, collected and analyzed by the controller, and then transferred to the "downstream" simulator. Such communication processes will continue until all the servers finish the simulations. Finally, the controller can output the simulation results as a whole. If multiple runs needed, the controller can also call the server simulators for the next run.

Example Problems and Test Environment

Two example networks were built to test the performance of the proposed distributed modeling framework. Example network 1 was coded based on the real network, but expand artificially for the computational load study. The origin real network is part of the Main Street of Logan, UT, including one main arterial, 10 signalized intersections and 22 O-D demand zones. We extended the network by double the links and O-D demand zones as shown in Figure 5(a). For simple analysis, there is only one connector between the two distributed main streets, and the demand matrix was also modified correspondingly. So there are totally 20 signalized intersections, 42 O-D demand zones in this test network. Since the network is so simple that without route choice, the all-or-nothing assignment method was applied here. Example Network 2, as shown in Figure 5(b), was artificially designed as a large-scale grid network. The original network is coded with 22 arterials, 120 signalized intersections, 44 O-D demand zones, and the peak hour demand of the whole network is 33840 vehicles. Two scenarios were tested with this grid network. In Scenario 1, the grid network was evenly divided by 2 parts and simulated at 2 clients with the proposed distributed modeling framework. So each network will be comprised by 16 arterials, 60 signalized intersections, and 32 O-D demand zones. The sub-network boundary includes 10 connections or transfer zones. In Scenario 2, the grid network was evenly divided by 3 parts with each sub-network

include 12 arterials, 40 signalized intersections, and 28 O-D demand zones, distributed simulated on 3 server simulators. Sub-networks on server 1 and 3 include 10 connections with server 2, and the sub-network on server 2 has 20 connections: 10 with server 1 and 10 with server 3.

Testing was carried on three desktops, the double Logan network and Scenario 1 of the grid network were carried at two desktops each with one 2.8 GHz CPU processor and 1 GB RAM. Scenario 2 of the grid network was carried with one additional workstation that has 3.2 GHz CPU processor and 2 GB RAM. All of the operation systems are Windows XP Professional. Each client computer was carried one part of the origin network in Paramics Modeler Version 4.21. All other traffic conditions and simulation configurations at each simulator are the same.

Computation Load Study

Before test the benefit of the distributed modeling framework, a scenario was designed to test the computation load of updating vehicle information. The Example network 1 was run both with the distributed simulation and non-distributed simulation scheme at different time-step, 2 per second till 8 per second. The demand matrix assigned in the network is zero or full demand. With zero demand simulation, there is no vehicle update in the network and the CPU time are all cost by controller update, network update, shortest path update, and etc. From Table 1, we can see that the network load occupied almost 30% of the whole demand assigned in the simulation network. The computation load for updating vehicle information is occupied near 70% of the CPU computation load. Such trend can also be verified from Table 2, where we list the CPU time contribution of updating vehicle information with ¼, ½ and full demand by decrease the CPU time with zero demand. It can be easily see that the computation time is also doubled when the demand is doubled. This study certificated that the total computational load for the microscopic traffic simulation is dominated by the number if the vehicles in the network.

Benefits from the Distributed Modeling Framework

Table 3 shows the speedup result of the Example Network 1 with the distributed modeling framework methodology proposed in this paper. As the synchronized time window increase, the speedup will increase from 0.73 up to 1.52 with time-step 2 per second and from 1.23 up to 1.65 with time-step 8 per second. We can also see that the speedup from distributed simulation will converged around 1.6 times with the time-step of 8 per second. Note that the pure computational load of 8 time steps per second is 4 times of that 2 time steps per second. However, given the fixed synchronization time window, the communication overhead is the same for both cases. That's why we achieve more benefits with higher simulation time steps. The status of time-step increase is quite similar with the status of the number of vehicle in simulation increase, so from this result we can conclude that we will get more benefit from the large-scale network comparing with the small size network, it will be verified from the results of the Example Network 2 below.

Figure 7 shows the speedup benefit of Example Network 2 with Scenario 1, the large-scale gird network was evenly divided by 2 parts and distributed simulated on two desktops. From the figure we can see that as the synchronized time window increase, we can achieve more benefits because of the communication overhead is decreased and the waiting times of the faster simulations are also decreased. However, the benefits from increasing the synchronized time window is not significant, which could be explained from the above computation load study that the dominant simulation time consuming is vehicle update. In this scenario, the speedup comparing with server 1 and server 2 are different. As we mentioned before, all the clients are run as fast as the slowest one, so the benefit from the proposed distributed modeling framework in this scenario was up to 2.4 with 2 per second time-step, and up to 2.7 with 8 per second time-step, which is higher than the benefits get form Example Network 1, a small size network.

Another scenario of the Example Network 2 was designed to divide the grid network into 3 parts and distributed at three computers. The benefit results comparing Scenario 1 and Scenario 2 are shown at Figure 7, both scenarios choose the speedup results from the slowest server. This figure shows the same trend as the previous examples. An additional conclusion is obviously that we will get more benefits with more distributed servers. However, due to the communication load will be significant increase as the number of clients increase, some benefits from the vehicle update will be counteracted. Which means the benefit form increasing the number of distributed client may be limited.

Consistency between Distributed and Sequential Simulations

One critical issue in the distributed simulation is the consistency, meaning that the distributed simulation has to be consistent with the sequential simulation in terms of simulated traffic conditions. In this paper, we only verify the consistency issue with the total number arrivals in each zone of both distributed and sequential simulation. Table 5 shows the number of vehicles that arrived to the destination zones in the distributed simulation comparing with the origin demand data. This is a 2-server distributed case data results, the data with Italic fonts and gray shading was generated from server 2, and the other part was the data from server 1. In Paramics, the user can request the simulator release the vehicles strictly according to the demand files. So in the table we can see that there is a very little difference between the two columns, and the total arrived number is exactly the same. Because the vehicle transferred does not carry its destination of the whole network, the destination is assigned according to the proportion of the arrival demand at the “downstream” simulators. Hence the total arrived number will have a little difference due to the randomness. The result shows in Table 5 verified the consistency of our proposed framework.

CONCLUDING REMARKS

A distributed modeling framework for large-scale network microscopic simulation is proposed in this paper. The methodology divide the large network into multiple parts and distributed simulate the sub-networks on multiple computers. Windows socket is

employed as the communication middle ware to transfer the synchronized time clock and vehicle information between the client controller and server simulators. Paramics Version 4 is selected as the simulation demonstration tools. Two example networks were designed to test the performance of the proposed distributed modeling framework. The computation load study shows that vehicle update is the domination computation cost of simulation. By increasing the time-step and comparing the benefit from the two examples, we can conclude that the methodology is suitable for large-scale network. The results also shows that more benefits will get by increasing the synchronized time window, but not significant. Another conclusion is that we could get more benefits by employed more processors, however, part of the benefit will be counteracted by the communication time consuming between the controller and simulators.

As mentioned before, the research demonstrated in this paper is a light global controller / impendent subnets, which is only the first part of this proposed distributed modeling framework. To achieve more accurate distributed simulation results, the heavy global controller / coordinated subnets design should be considered in the further research, including develop the algorithm for global routing, network decomposition, and etc. These issues will be explored in the subsequent paper.

REFERENCES

(1) Moore, G. E. Cramming More Components onto Integrated Circuits. *Electronics, Vol. 38, No. 8, April 19*, 1965.

(2) Lee, Der-Horng and Chandrasekar P. A Framework for Parallel Traffic Simulation Using Multiple Instancing of A Simulation Program, *Intelligent Transportation Systems, Vol. 7, No. 3-4, pp. 279-294*. 2002,

(3) Bernauer, E. Breheret, L., Algers, S., Boero, M., Taranto, C. D., Dougherry, M., Fox, K. and Gabard, J. F., Review of Micro-Simulation Models Appendix D., Ref: SMARTEST/D3, Institute of Transportation Studies, Leeds, U.K., Universitu of Leeds, 1998.

(4) Barceló, J, Ferrer, J.L, García D., Florian, M. and E. Le Saux, The Distributedization of AIMSUN2 Microscopic Simulator for ITS Applications, *Proc. 3rd. World Congress on Intelligent Transport Systems, Orlando*, 1996.

(5) Cameron, G. and Duncan, G., PARAMICS-Distributed Microscopic Simulation of Road Traffic, *The Journal of Supercomputing, Vol.10, pp.25-53*, 1996.

(6) Nagel, K., and Rickert, M., Dynamic Traffic Assignment on Parallel Computers in TRANSIMS, *Future generation computer systems, Vol. 17, Issue 5, pp 637-648*, 2001

(7) Quadstone Limited, Paramics User Guide Version 4.0, *Quadstone Limited*, Edinburgh, UK, 2003.

(8) Microsoft Corporation. Distributed Component Object Model Protocol-DCOM/1.0, draft, November 1996. *Avalable: http://www.microsoft.com/Com/resources/comdocs.asp*

(9) Henning, Michi and Vinoski, Steve, Advanced CORBA Programming with C++, *Addison-Wesley Professional*, 1999.

(10) Microsoft Corporation. MSDN Library Online. *Avalable:http://msdn.microsoft.com/library/en-us/winsock/winsock/windows_sockets_start_page_2.asp*

(11) Ferscha A. Parallel and distributed simulation of discrete event systems. In Handbook of Parallel and distributed Computing, McGraw-Hill, 1995.

TABLE 1 Computation Load percentage of those other than Updating Vehicle Information.

	Time-step (per sec)	**Simulation Time (sec)**		**%**	**Demand (veh/hr)**
		Zero Demand	**Full Demand**		
Distributed Simulation	2	7	21	33	6813
	4	12	38	32	
	8	21	73	29	
Non-Distributed Simulation	2	11	41	27	13294
	4	21	77	27	
	8	35	131	26	

TABLE 2 Computation Load Contribution of Updating Vehicle Information.

	Time-step (per sec)	**Simulation Time / Vehicle Contribution (sec)**			
		Zero Demand	**¼ Demand**	**½ Demand**	**Full Demand**
Distributed Simulation	2	7 —	11 4	14 7	21 14
	4	12 —	20 8	25 13	38 26
	8	21 —	35 14	46 25	73 52
Non-Distributed Simulation	2	11 —	19 8	25 14	41 30
	4	21 —	34 13	45 24	77 56
	8	35 —	62 27	85 50	137 102

TABLE 3 Speedup of the Example Network 1.

Time-step (per sec)	**Synchronized Time Clock (sec)**	**Distributed Simulation (sec)**	**Non-Distributed Simulation (sec)**	**Speedup**
2	10	56	41	0.73
	20	39		1.05
	30	33		1.24
	40	30		1.37
	50	29		1.41
	60	27		1.52
8	10	111	137	1.23
	20	94		1.46
	30	88		1.56
	40	84		1.63
	50	84		1.63
	60	83		1.65

TABLE 4 Arrival number of Vehicles Comparing with the Original Demand with a 2-Server for Grid Network.

Zone	Demand	Arrived	Diff.	Zone	Demand	Arrived	Diff.
1	710	712	2	21	470	472	2
2	730	720	-10	22	690	690	0
3	750	754	4	23	710	714	4
4	770	763	-7	24	730	732	2
5	790	789	-1	25	750	739	-11
6	810	808	-2	26	770	776	6
7	830	828	-2	*27*	*790*	*797*	*7*
8	850	863	13	*28*	*810*	*804*	*-6*
9	870	882	12	*29*	*830*	*821*	*-9*
10	650	642	-8	*30*	*850*	*840*	*-10*
11	*650*	*657*	*7*	*31*	*870*	*879*	*9*
12	*870*	*870*	*0*	*32*	*890*	*882*	*-8*
13	*850*	*851*	*1*	33	890	882	-8
14	*830*	*833*	*3*	34	870	877	7
15	*810*	*823*	*13*	35	850	842	-8
16	*790*	*794*	*4*	36	830	830	0
17	*770*	*784*	*14*	37	810	818	8
18	*750*	*741*	*-9*	38	790	787	-3
19	*730*	*728*	*-2*	*39*	*770*	*762*	*-8*
20	*710*	*714*	*4*	*40*	*750*	*743*	*-7*
				41	*730*	*723*	*-7*
				42	*710*	*713*	*3*
				43	*690*	*684*	*-6*
				44	*470*	*477*	*7*
				Total	**33840**	**33840**	**0**

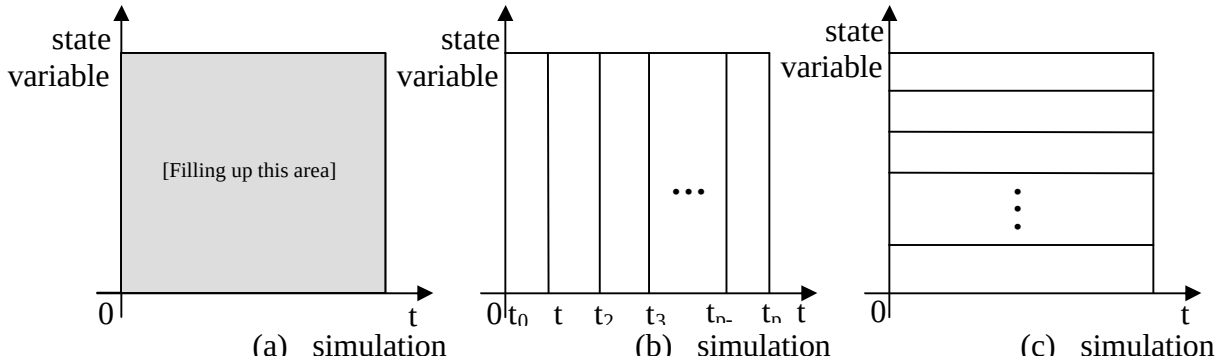

Figure 1: (a) The space-time graph. Simulation execution, which runs from simulated time *0* to time *t*, is equivalent of filling in the shaded area. Possible decomposition with respect to (b) space domain and (c) time domain.

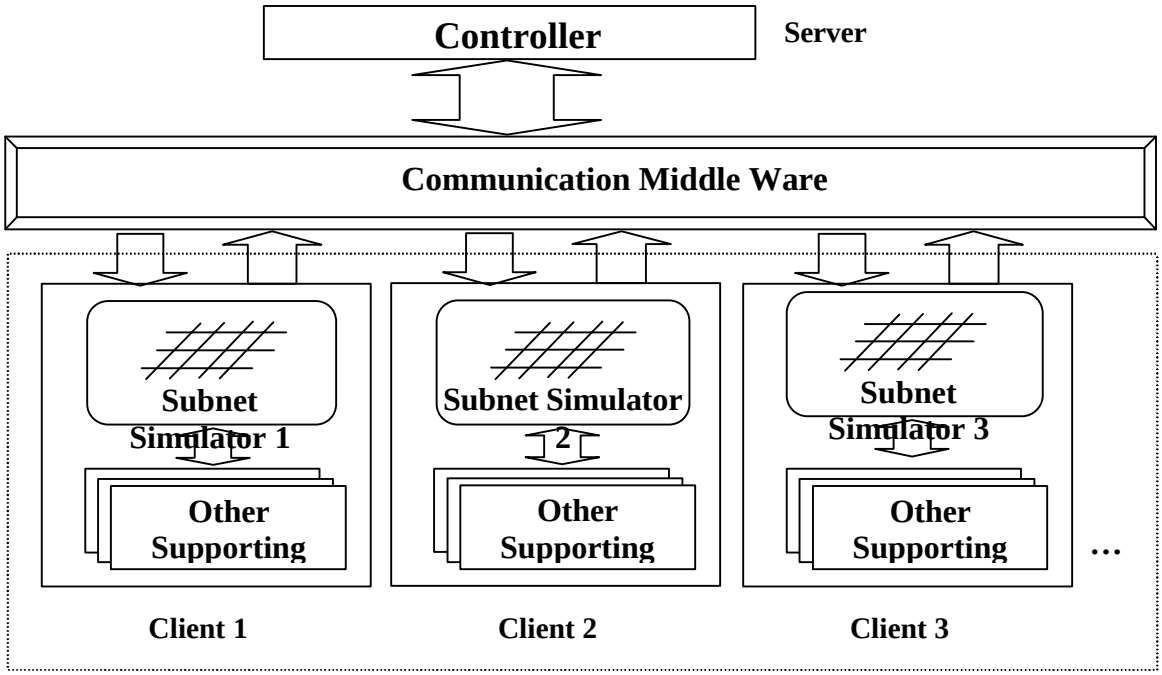

FIGURE 2 General Distributed Modeling Architecture.

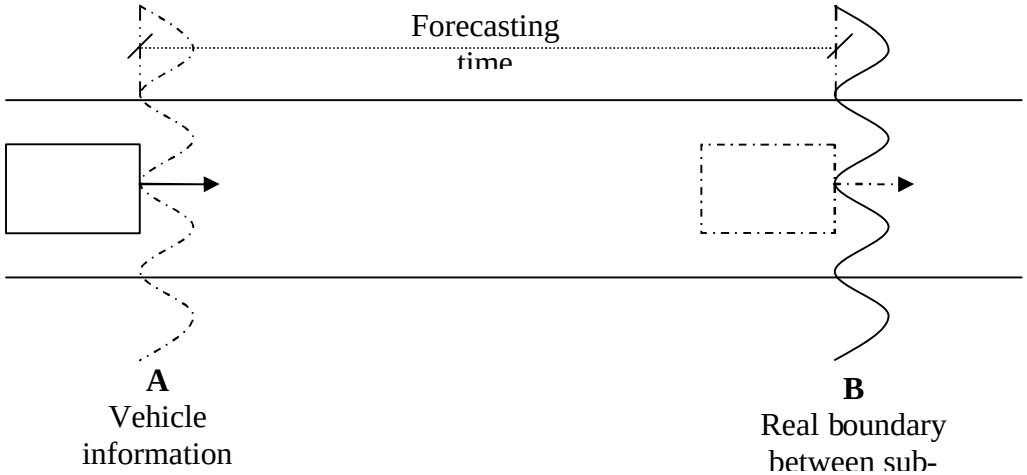

FIGURE 3 Transfer Vehicle Forecasting.

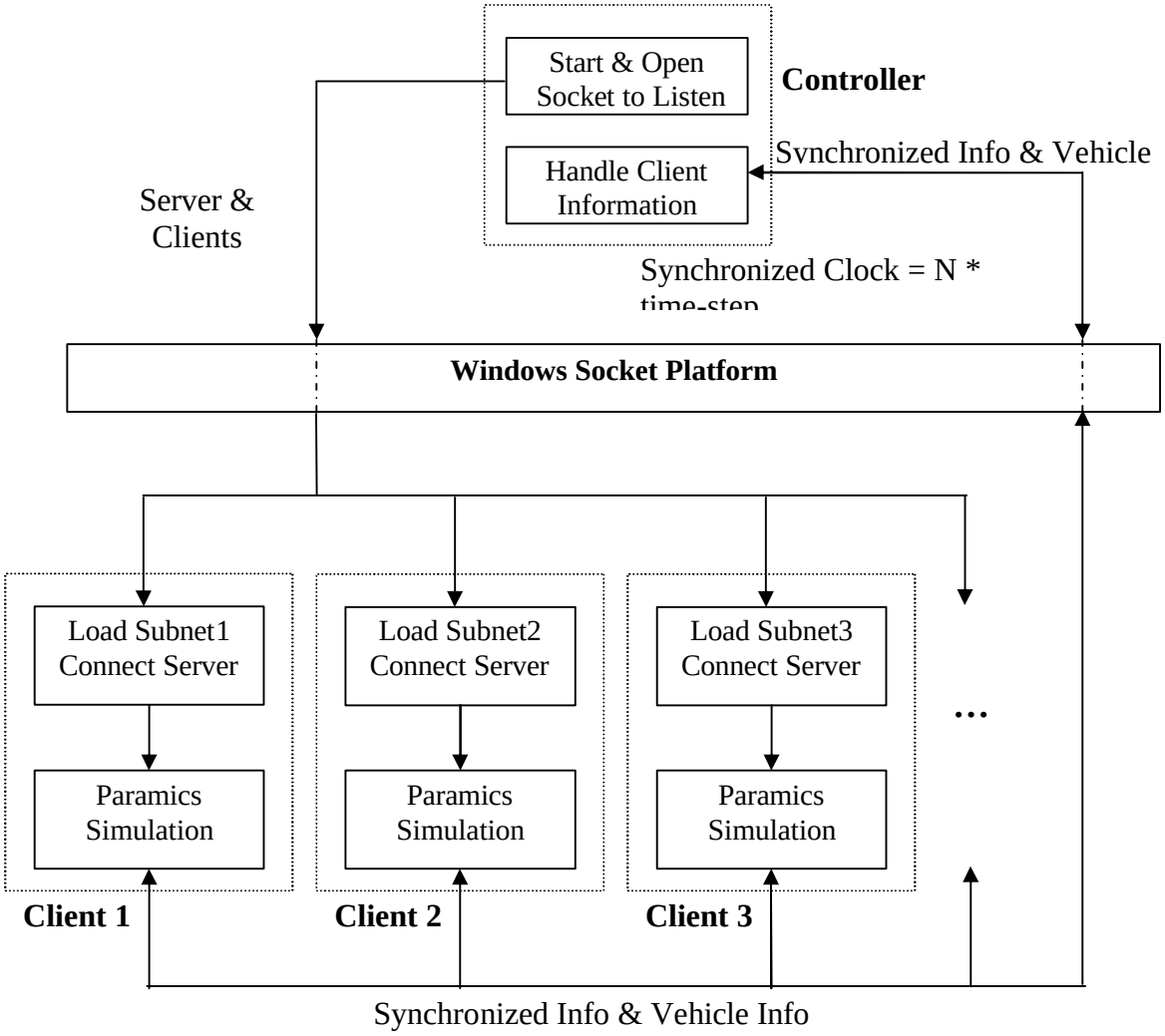

FIGURE 4 Prototype of the Distributed Modeling Framework.

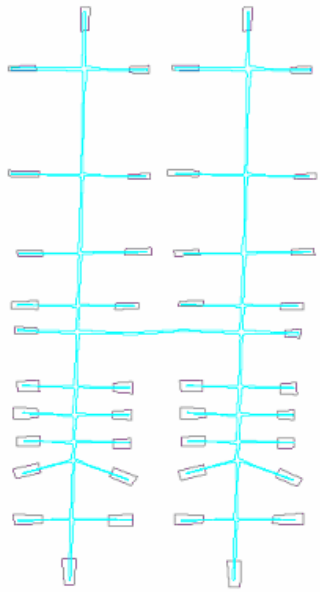

FIGURE 5(a) Example network 1 — Double Logan network.

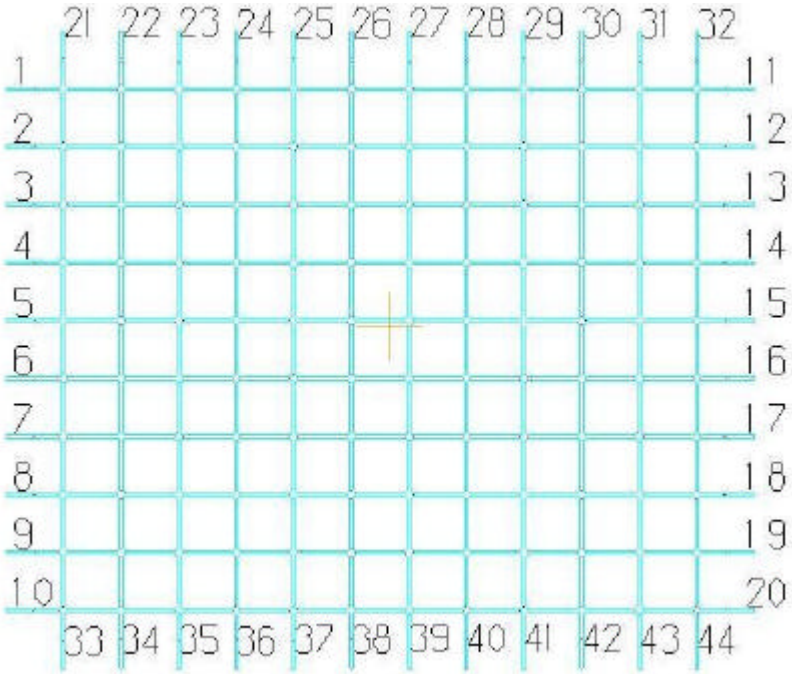

FIGURE 5(b) Example network 2 — Large-scale grid network.

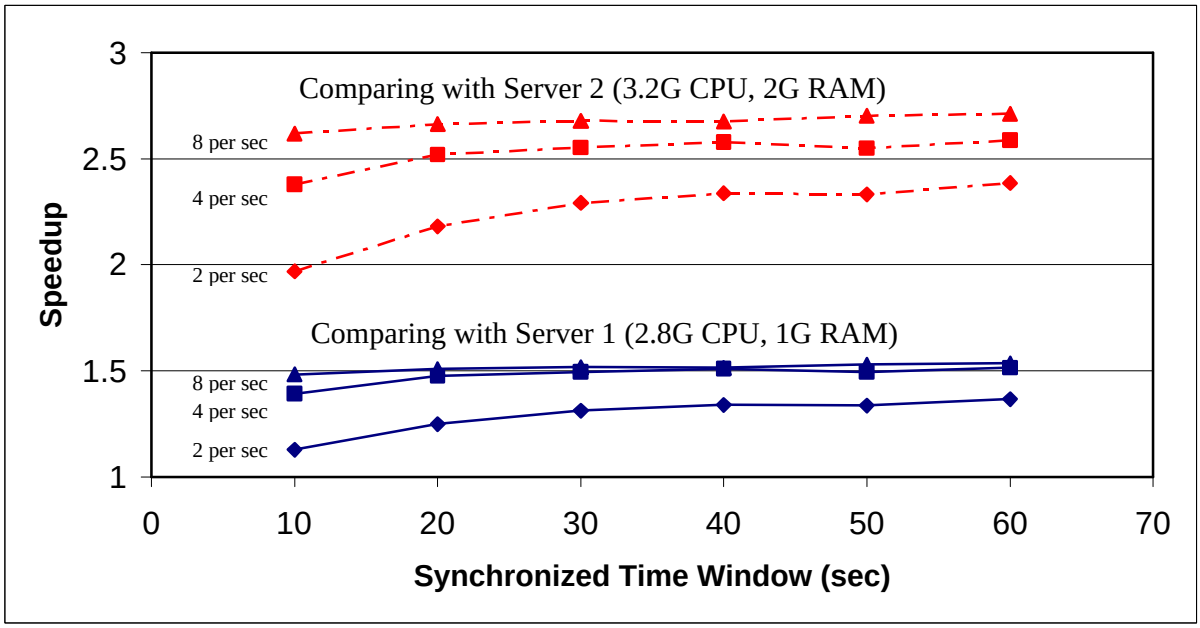

FIGURE 6 Speedup of the Example Network 2 with Scenario 1.

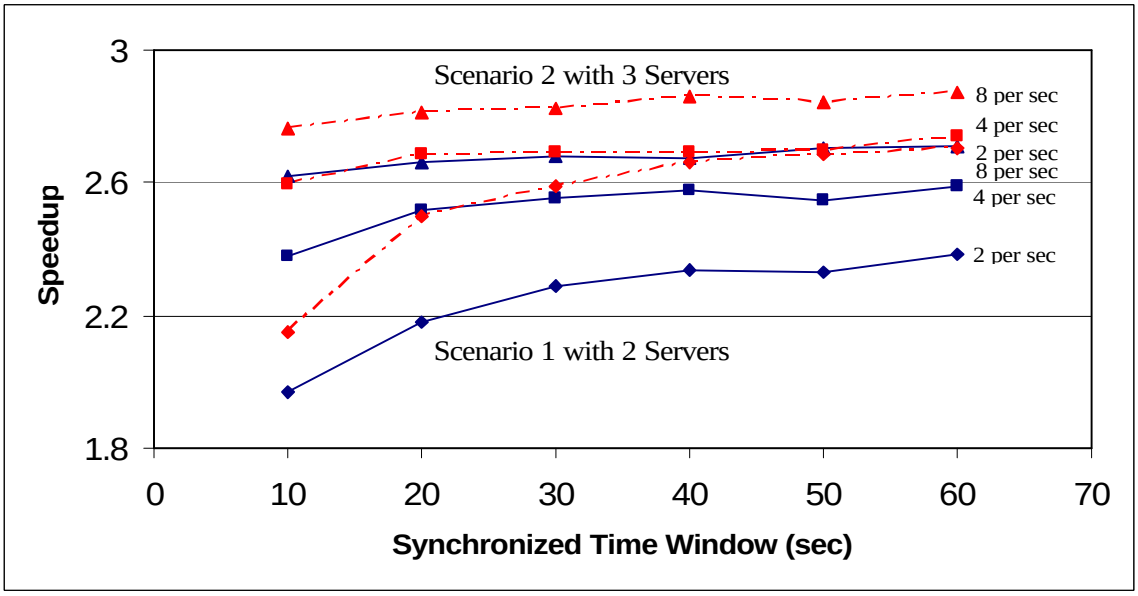

FIGURE 7 Speedup comparing with Scenario 1 and Scenario 2 of the Example Network 2.