An Erlang-based Simulation Approach of Artificial Transportation Systems*

Songhang Chen, Fenghua Zhu, *Member, IEEE*, and Fei-Yue Wang, *Fellow, IEEE*

***Abstract*—Artificial Transportation Systems (ATS) can support a variety of computational experiments for different purposes, and have become important tools for transportation research. However, when the road network modeled in ATS is large or there are a lot of vehicles, ATS will run very slow or even cannot be started due to the memory limit of a standalone computer. With the acceleration of urbanization process and rapid increase of car ownership, it is necessary to find a high-performance computing method for running large-scale ATS. Therefore, the paper presents an Erlang-based approach to realize concurrent and distributed computing of ATS. To verify the feasibility, a prototype is built, and a performance test is conducted. The results show the method can achieve the efficiency utilization of computing resources.**

I. Introduction

The urban transportation system is a kind of "natural" multi-disciplinary complex giant system [1]. With the acceleration of urbanization process and rapid increase of car ownership, such as in China, the size of urban transportation system grows larger, and the complexity of its interaction with population, nature and society increases quickly. These bring stern challenges to traditional transportation simulation systems, so Wang *et al.* put forward the concept of Artificial Transportation Systems (ATS) [2][3].

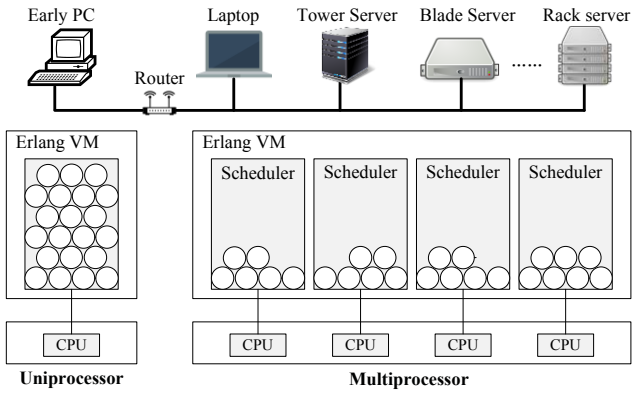
ATS adopt a new strategy for transportation simulation. Typically, it uses object-oriented program method to model each participator in the real transportation system as an agent with a certain autonomy, sociality, proactivity and mobility. The method is good at describing the intuition, experience knowledge, behaviors of peoples as well as interaction among them. However, it also means that ATS require more computation. Especially, when the road network modeled in ATS is large or there are a lot of vehicles, ATS run very slow or even cannot be started due to the memory limit of a standalone computer.

The similar thing happened on traditional microscope transportation simulation. It has motivated some researchers to study distributed transportation simulation [4][5]. The basic thought is native: divide a big road network into small ones, and then distribute them across multiple low-cost computers to simulate coordinately. The method is more economical than using high performance computer systems like mainframe computers or parallel computers by far. Following the idea, we raised and implemented a P2P framework to achieve distributed simulation of ATS in [6]. The performance test of this framework shows that it can offer more computing ability than standalone simulation and can support large-scale simulation. In this way, different areas of the road network can be simulated concurrently, but transportation simulation in each area is still sequential, and the synchronization mechanism among areas also limits the performance. Therefore, we started to search an improved way, and Erlang attracted our attention.

The remaining of this paper is organized as follows: Section II presents an introduction of Erlang; Section III addresses a typical mechanism of ATS and its implementation with Erlang; Section IV describes and tests a prototype developed with the proposed method; Section V concludes the paper.

II. Introduction of Erlang

Erlang is a programming language used to build massively scalable soft real-time systems with requirements on high availability. It was originally a proprietary language within Ericsson in 1986 [7], but was released as open source in 1998 and widely used in telecom, banking, e-commerce, computer telephony and instant messaging. In most languages threads require external library support, while Erlang has a small but powerful set of primitives to create processes and communicate among them. They are neither operating system processes nor operating system threads, but lightweight processes that are scheduled by Erlang Virtual Machine (Erlang VM). The only way for processes to interact is through message passing [8]. This makes concurrent and distribution computing easier, and also makes Erlang more appropriate to current computing environment with multicore and cloud architecture.



*Research supported by the Early Career Development Award of SKLMCCS (99S9021F4U) and partly by National Natural Science Foundation of China projects (71232006, 61233001, 61533019, and 91520301).

Songhang Chen, Fenghua Zhu and Fei-Yue Wang are with The State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China and Qingdao Academy of Intelligent Industries, Qingdao, Shandong, 266109, China (songhang.chen@ia.ac.cn; fenghua.zhu@ia.ac.cn; feiyue@ieee.org).

Figure 1. Erlang processes running on uniprocessor and multiprocessor hardware respectively. Erlang VM automatically distributes the workload over the available CPU resources.

Especially, Erlang can deal with the physical execution of tasks automatically. As illustrated in Fig. 1, if extra CPUs are available, Erlang uses them to run more of concurrent tasks in parallel. If not, Erlang uses what CPU power there is to do them all a bit at a time. Users do not need to think about such details, and the Erlang programs automatically adapt to different hardware. They just run more efficiently if there are more CPUs, as long as the software engineers have tasks lined up that can be done concurrently [9]. Erlang's application can be executed either on single-core, multicore and distributed computing architectures. Moreover, benefit from Erlang VM, the application owns the feature of "write once run everywhere" like Java.

Until now, to our knowledge, there is still not much work on using Erlang in simulation. Luo *et al.* applied Erlang in the simulation of electric vehicle charging infrastructure to deal with the real-time power sum generated during the charging and ensure the sum have less distortion at the same time [10]. Toscano *et al.* describe a concurrent simulation middleware implemented in Erlang to improve the efficiency of discrete event simulation [11].

III. ERLANG-BASED SIMULATION OF ATS

A. The Mechanism of ATS

ATS can be treated as an extension of traditional microscopic transportation simulation (TMTS) from the perspective of complex systems [12]. The traffic flow they display may be similar, but the principles behind are very different. TMTS is usually based on initial static or dynamic Origin-Destination matrix, while ATS is typically based on the theory of traffic demand generation based on activity (TDGA) [13]. Concretely, every traffic participant is modeled as an agent with certain autonomy, and a bottom-up mechanism is designed to generate travel behaviors as shown in Fig.2.

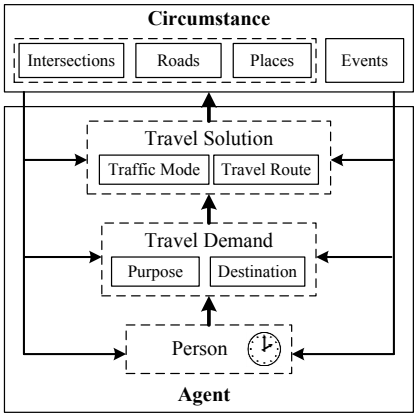


Figure 2. Generate person's travel activity from bottom up.

Firstly, agents are enabled to check the system clock actively. If it is not the end time of the present activity, it has to stay in the current place. Otherwise, it will use the Travel Demand Model (TDM) to select demand and destination of next travel. Then, through the Travel Solution Model (TSM), a travel solution comprised of one or multiple pairs of route and mode can be decided. Thirdly, agents will start to move in the road network according to the solution. Finally, after arriving at the destination, agents will invoke the Activity Duration Model (ADM) to decide how long it should stay in the destination to finish travel demand.

Besides the road network which is mainly composed of intersections, roads and places, the circumstance in ATS also includes abstract natural or social process, which is composed of different events, such as weather, road work, traffic accident, and large-scale activity. By rules, the circumstance takes effect on behaviors of agents [12]. For example, if it is rainy or foggy, some types of travel demand will be weakened, and agents on roads will decrease driving speed with a certain probability for safety.

B. Process Infrastructure

Erlang adapts for such bottom-up idea well. In Erlang, each agent can be coded as a process and act concurrently as in the real world. Without loss of generality, a minimal implementation of such ATS can be abstracted as the following five software modules.

(1) *Road Network Module*: hold the geometric data of road network and display dynamic traffic scenes, including the movement of vehicles, bicycles, and pedestrian.

(2) *Population Module*: responsible for generating all agents in ATS and schedule them to make behaviors.

(3) *Behavior Module*: responsible for deciding all people's behaviors, including activity behavior and travel behavior.

(4) *Statistics Module*: gather statistics on data generated during the running of ATS.

(5) *Time Module*: control the start time, step size, and end time of ATS simulation.

Following the programming paradigm of Erlang, we divided these five modules into three kinds of processes, which together constitute a process tree of 1 : N : M as shown in Fig. 3.

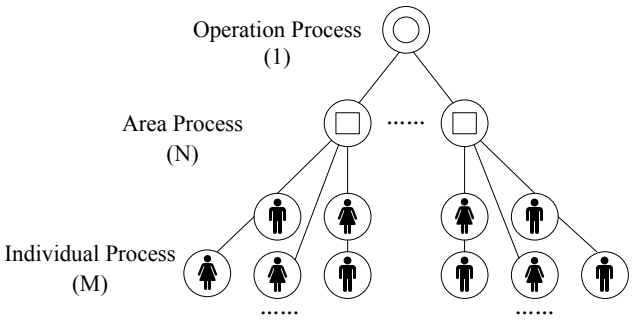


Figure 3. The process infrastucture of Erlang-based ATS.

(1) *Operation Process*: control the whole simulation of ATS, including initialization, start, pause, and termination. The process replaces the role of *time module*. Before simulation, an operation process must be started firstly.

(2) *Area Process*: correspond to an area of the road network modeled in ATS. Its responsibility is to generate individual

processes and gather regional transportation statistics. The process contains the role of *statistics module*.

(3) *Individual Process*: correspond to each individual in *population module*. Models like TDM, TSM and ADM can be integrated in this process. The process integrates the role of *behavior module*.

Meanwhile, a Mnesia-based distributed database is built to hold the data in *road network module* and *population module*. Mnesia is a distributed, soft real-time database management system written in Erlang [14]. It can provide data service for three kinds of processes either on a standalone computer or distributed computing architectures.

C. Information Infrastructure

Generally speaking, communication and synchronization between different subtasks are the greatest obstacles to getting good performance in distributed computing. Therefore, it is quite important to design interactions among the three kinds of processes. Erlang processes only can interact by messages, so we design an information infrastructure as shown in Fig. 4.

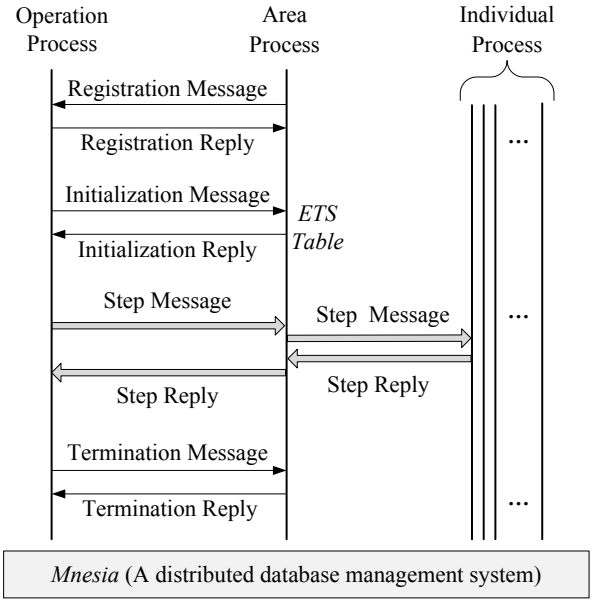


Figure 4. The information infrastucture of Erlang-based ATS.

There are four kinds of messages between an operation process and an area process.

(1) *Registration Message*: After an area process is started, it will send a registration request to the operation process by this message. The message includes information about local computing resources, so that the operation process could allocate simulation tasks according to computing ability.

(2) *Initialization Message*: The message is used to allocate simulation tasks among area processes. When an area process receives the instruction, it will acquire related data from the distributed database by the area ID in the message. Then, according to the population data within, the area process will create individual processes.

(3) *Step Message*: Every simulation step, the operation process sends the message to all registered areas processes, and only when it receives all of their replies, it can send the next one. On the other hand, when an area process receives a *step message*, it transmits the message to all created individual processes, and does not relay to the operation process until all individual processes reply. By the *step message*, the operation process controls the progress of ATS simulation.

(4) *Termination Message*: The message is used to notify area processes to exit and then terminate the whole ATS halfway.

It must be noted that an area process will create a public ETS table, which is an efficient in-memory key-value database included with the Erlang VM, to store the initial status of all individuals after started. When an individual process receives a *step message*, it will look up the ETS table to get requisite information firstly, and then decide and make behaviors, and finally reply messages and update the ETS table. Individuals with different status need different information. For example, when an individual drives on a road, it will need the information about vehicles around and traffic lights forward; when an individual will drive out of the current area, it will need the process information of the adjacent area; when an individual do an activity at a place, it will need the information of simulation time to trigger the next activity. Traffic-related statistics are also gathered based on this ETS table and stored into the distributed database. Since an area process and individual processes which it creates are on the same machine, their communication is very fast.

IV. A Prototype

To verify the feasibility of the proposed method, we built a prototype, which makes some simplifications but covers all message interactions, and test its performance.

A. Experiment Setting

First of all, an experimental ATS must be constructed. We choose a 3×4 grid road network as transportation simulation area, and the number of artificial population is set to 240,000, and the simulation period is set from 6:00 to 22:00 which covers the morning and evening rush hour of traffic. Each simulation step size represents 1 second in actual. Table I shows the hardware configuration of machines used in experiments. The machines are connected by local area network, the maximal data transfer rate of which reaches 100.0Mbps.

TABLE I: Hardware configuration of nodes

Node No.	CPU Count	Operation System	Main Memory
#1	1	Windows 7 32bit	1.00G
#2	2	Windows 7 32bit	4.00G
#3	4	Windows 7 32bit	4.00G
#4	4	Windows 7 64bit	4.00G
#5	4	Windows 7 64bit	4.00G

B. Preformation Testing

To measure and compare performance, we define speedup $S(p)$ and efficiency $E(p)$ for Erlang-based ATS simulation as following:

$$S(p) = \frac{T(n,1)}{T(n,p)} \quad (1)$$

$$E(p) = \frac{T(n,1)}{pT(n,p)} = \frac{S(p)}{p} \quad (2)$$

where $T(n,p)$ is the run-time of simulation executed on p processors. Speedup is the improvement in speed of execution of a task on two systems. Linear speedup or ideal speedup is obtained when $S(p) = p$. Efficiency is a metric of the utilization of the resources of the improved system, and its value is typically between 0 and 1. Programs with linear speedup or running on a single processor have an efficiency of 1 [15].

We design and conduct five groups of experiments with different nodes as noted in Table II. The Exp.1, 2, and 3 are designed to test the performance of Erlang-based ATS simulation in multicore computing architectures, and the Exp.4 and 5 are carried out to test the performance in distributed computing architectures. Each experiment is repeated five times to get an average run-time. The changes of speedup and efficiency with CPU counts are shown in Fig.5.

TABLE II: RESULTS OF PREFORMATION TESTING

Exp. No.	Selected Nodes	Average Run-time	Speedup	Efficiency
1	#1	38.09h	1	1
2	#2	25.00h	1.52	0.76
3	#3	18.18h	2.09	0.52
4	#3、#4	9.88h	3.86	0.48
5	#3、#4、#5	6.37h	5.98	0.50

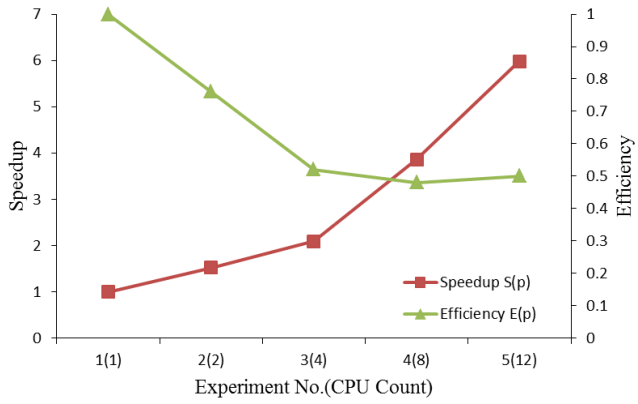


Figure 5. The result of performance test.

C. Result Discuss

The simulation period is over 16 hours (from 6:00 to 22:00), while the average run-time of Exp.1 exceeds 38 hours. Such a long time indicates that the Node #1 with a single CPU is overloaded. Node #2 and #3 has two and four CPUs respectively. Comparing Exp.2 and 3 with Exp.1, we can find that the speedup $S(p)$ increases with the CPU count on the standalone computer (but is far from linear speedup), and the efficiency $E(p)$ decreases. The average run-time of Exp.3 exceeds 18 hours, which is still longer than the simulation period 16 hours. This indicates that though the simulation task is speeded up by multicores, it also overloads Node #2 and #3.

In Exp.4 and 5, the simulation is deployed in distributed computing architectures. Respectively, the 3×4 grid road network is divided into two 3×2 areas and three 1×4 areas, the size of which is about the same. Since the generated traffic flow is distributed uniformly over the whole road network, such division can ensure load balance to a certain extent. Comparing Exp.4 and 5with Exp.3, we can find that the speedup $S(p)$ increases with node count quickly, and the efficiency $E(p)$ has no obvious changes. This indicates that the distributed computing architecture of Experiment 4 and 5 still can support more large-scale simulation, and meanwhile more nodes can speed up the simulation further.

Given that the machines used in experiments are common personal computers, it can be expected that the performance test will be more accurate when deployed in a cloud or cluster platform having good consistency.

V. CONCLUSION

The paper proposes a novel Erlang-based approach to build ATS. Every participant in the actual transportation system is represented as a kind of lightweight process which only can interact with others by messages. In this way, the concurrent and distributed computing of ATS becomes an easy and native thing. A prototype is implemented to verify the method, and the performance test shows that the new ATS can achieve efficiency utilization of computing resources.

Future research should focus on further replenishing the prototype. How to ensure load balances during distributed simulation must be studied. This problem is essentially equal to how to divide a road network into areas according to given computing resources.

ACKNOWLEDGMENT

The authors would like to thank those people that contributed to this work in The State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences.

REFERENCES

[1] F.-Y. Wang, S. M. Tang, "Concept and Framework of Artificial Transportation Systems," *Journal of Complex Systems and Complexity Science*, 2004, 1(2):52-59.

[2] F.-Y. Wang, "Integrated Intelligent Control and Management for Urban Traffic Systems," *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, Oct. 2003, Shanghai: 1313-1317.

[3] F.-Y. Wang, S. M. Tang, "A Framework for Artificial Transportation Systems: From Computer Simulations to Computational Experiments," Proceedings *of 8th IEEE International Conference on Intelligent Transportation Systems*. Sep. 13-16 2005, Vienna, Austria: 1130-1134.

[4] R. Klefstad, Y. Zhang, M. Lai, "A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation," *Intelligent Transportation System*, 2005, 6:813-818.

[5] B.G. Ma, W.F. Wang, C.J. Hou, F.Z. Qian, "Realization of .NET Remoting-based Distributed System," *Computer Technology and Development*, 2006.5, 16(3):50-55.

[6] S.H. Chen, S.M. Tang, F.H. Zhu, Q.H. Miao, "Research on the Distributed Framework for Computational Experiments Based on Artificial Transportation Systems," *Journal of System Simulation*, 2013, 25(4), 605-611.

[7] J. Armstrong, "History of Erlang," *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, 2007.

[8] J. Armstrong, *Programming Erlang: Software for a Concurrent World*, Pragmatic Programmers, USA, 2013.

[9] M. Logan, E. Merritt, R. Carlsson, *Erlang and OTP in Action*, Manning, 2011.

[10] H. Luo, Z. Li, M. Ding, "Electric vehicle Charging for electric facilities simulation based on Erlang," *Sciencepaper Online*, 2014.

[11] L. Toscano, G. D'Angelo, M. Marzolla, "Parallel discrete event simulation with Erlang," *Proceedings of the 1st ACM SIGPLAN workshop on Functional high-performance computing*, 2012, ACM, New York, NY, USA, 83-92.

[12] J.Y. Li, S.M. Tang, X.Q. Wang, F.-Y. Wang, "Growing Artificial Transportation Systems: A Rule-Based Iterative Design Process," *IEEE Transaction on Intelligent Transportation System*, 2011, 12(2): 322-332.

[13] W. Davidson, R. Donnelly, P. Vovsha, "Synthesis of first practices and operational research approaches in activity-based travel demand modeling," *Transportation Research Part A: Policy and Practice*, 2007, 41: 464-488.

[14] H. Mattsson, H. Nilsson, C. Wikström, "Mnesia—A distributed robust DBMS for telecommunications applications," *International Symposium on Practical Aspects of Declarative Languages.* Springer Berlin Heidelberg, 1999: 152-163.

[15] I. Singh, "Review on Parallel and Distributed Computing", *Sch. J. Eng. Tech.*, 2013, 1(4):218-225.