MATMASSim: Multi-Agent Transportation Simulation Using MASS

Zhiyuan Ma

University of Washington, Bothell

Computing & Software Systems

Contents

Introduction

Traffic simulation has always been a difficult task. Each person want to have access to transportation without bothering by it. And each person possess their own free will to act and behave, which make the simulation tougher because it is impossible to predict beforehand. From this point, any software designed to provide a solution for this needs to address the fact that people are "intelligent", which means they are capable and expected to adapt and to learn [1].

A good approach to such complex problems is using multi-agent simulations. Instead of simulate with traditional mathematical or traffic flow models which just apply to the input data statically, multi-agent simulations provide a dynamic environment in which each individual agent, in particular the travelers will keep taking actions based on their internal rules. Therefore, this approach can simulate scenarios much more similar to those in the real world.

The rest of this chapter will introduce the two main aspects on which agent-based model have influence on: Traffic Assignment and Traffic flow simulation, as well as related concepts and theories. And finally we conclude the current parallelization techniques and their bottlenecks on performance.

I. Traffic assignment

Traffic assignment, also called route assignment or route choice, is a concept of route selection. Traffic assignment can be seen as the very essential part for transportation simulation, for which it is responsible for distributing and calculating traffic demand. Typically, traffic assignment can be categorized into two kinds: static and dynamic. In static traffic assignment, for a specific link within a network, its traffic flow parameters, such as inflow and outflow, are considered as constant or static, which means that it would remain the same value during the entire simulation, usually for one day. Therefore, its implementation is quite simple: just start with initial routes data, and run several iterations until it achieves User Equilibrium (UE), as Figure 1 shown below.

Algorithm 1 Macroscopic and static route assignment

1. **Initial conditions:** Compute some initial routes (e.g., best path on empty network for every OD pair).
2. **Iterations:** Repeat the following many times.
   (a) **Network loading:** Load the demand on the network along its routes and obtain network delays (congestion).
   (b) **Choice set generation:** Compute new routes based on the network delays.
   (c) **Choice:** Distribute the demand between the routes based on the network delays.

Figure 1. Static traffic assignment algorithm [2]

Although it might be suitable for transport planning for very long time period, such as 10 to 20 years, this static model is not very suitable to simulate short time period scenarios in real world. A very simple

example would be that traffic flow will fluctuate between different time period, and within the city network, certain links' flow will be much higher during peak hours compared to that in the midnight.

Therefore, dynamic traffic assignment (DTA) [3] was created in order to deal with this problem. For each origin-destination (OD) pair, its departure time is used by algorithm of dynamic model to include the consideration of traffic flow changes during different time blocks [4]. As Figure 2 shown below, it adds the parameter of departure time in both initial conditions and part (a) in iterations compared to the previous static traffic assignment algorithm.

Algorithm 2 Macroscopic and dynamic route assignment

1. **Initial conditions:** Compute some initial routing (e.g., best path on empty network for every OD pair and departure time).
2. **Iterations:** Repeat the following many times.
   (a) **Network loading:** Load all demand items on the network according to their departure times, let them follow their routes, and obtain network delays (congestion).
   (b) **Choice set generation:** Compute new routes based on the network delays.
   (c) **Choice:** Distribute the demand between the routes based on the network delays.

Figure 2. Dynamic traffic assignment algorithm [2]

Although DTA is much closer to reality by adding the traffic flow change during multiple time periods, it is still could not be seem as equal as reality, because traffic flow for each network during one specific time period would still remain the same, also it remain as an aggregate method for traffic assignment.

Finally, a disaggregate method is adopted, which is the agent based model. Within this model, traffic is no more assigned in a central manner; instead, it will be distributed to each traveler or agent, and traffic flow will be reflected in the network by collecting data from every traveler, as the algorithm in Figure 3 shown below. Also I sense the title of these three algorithms should be noted. In previous twos, they both says macroscopic in their algorithm title, which means from simulation's perspective, those two cannot be seem as detailed as microscopic method, which appears in the title of agent based algorithm. Further compared with DTA, since it is a totally different perspective in terms of the process of iterations, the agent based model is no more limited to change only between multiple time period; instead, each individual's behavior will affect the traffic flow as that traveler perform an action, such as changing route, leaving link, and etc. Hence, agent based traffic assignment model can be seem as the most equivalent to the traffic condition in reality.

Algorithm 3 Microscopic and dynamic route assignment

1. **Initial conditions:** Compute some initial routing (e.g., best path on empty network for every traveler).
2. **Iterations:** Repeat the following many times.
   (a) **Network loading:** Load all travelers on the network according to their departure times, let them follow their routes, and obtain network delays (congestion).
   (b) **Choice set generation:** Compute new routes based on the network delays.
   (c) **Choice:** Assign every traveler to a route (which can be the previously chosen one) based on the network delays.

Figure 3. Agent based model algorithm [2]

II. Traffic flow simulation model

As I become familiar with simulation structure, I tend to shift my focus to traffic model on flow micro-simulation, which is basically same to multi-agent based simulation. As I read through the part before testing, it's like reading the evolution of all the models, from the very beginning continuous model, to queue-based, to event-driven queue-based, and finally to a more improved dynamic event-driven queue-based approach. So basically, the continuous model is quite easy to implement, however, it could lead to huge amount of CPU resources waste since most of the agents could be just idle but still taking up CPUs' time. Therefore, event-driven based approach obviously seems to be a better solution. Yet, with this approach, the system have to add two other module, which are schedule and timer to deal with synchronization between huge number and agents and environments. As Figure 3 shows, more communication between scheduler, agents and timer is needed for implementation [5].

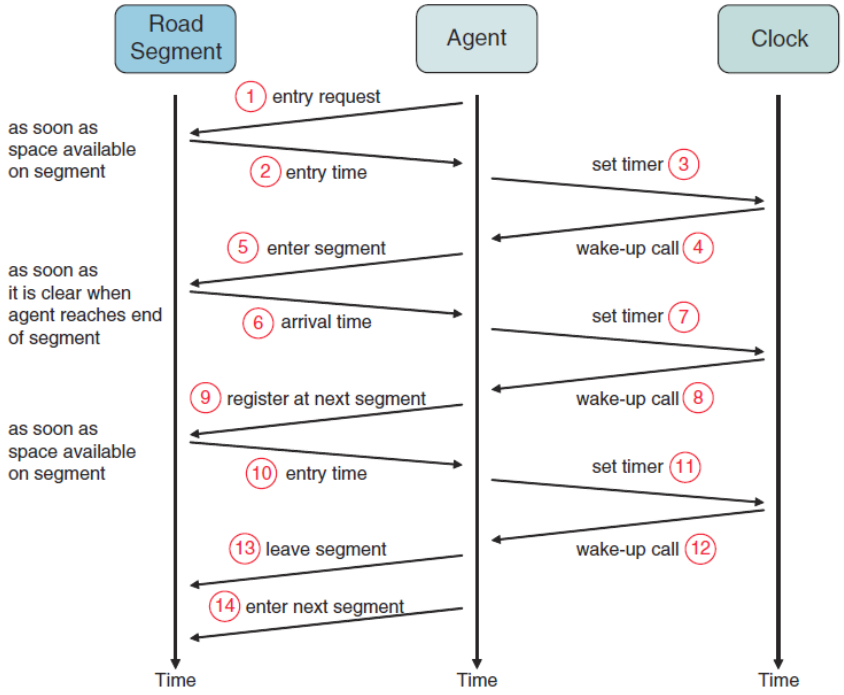

Figure 4. Event-driven queue based model control flow [5]

From another perspective, traffic flow simulation model can also be categorized by starting with physical micro simulation model, which are the most accurate and expensive type, including human car following, lane changing behavior. However, these methods are too expensive and not feasible for large scenarios. Then, a model called Cellular Automata is introduced, in which roads are divided into cells and each cell can be either empty or occupied by a car. However, this model become impractical for large numbers (>1 million) of cars, because it required update of each car's position in every time-step. Therefore, the Queue-Based model is created to solve this problem by representing links as queues. Hence, the queue-based model take this advantage and shift the computation from cars to links, and increase the simulation performance by a factor of 10 to 100 [6]. Because there is a high possibility that within one time-step, most of the agents don't require any action, which also means that they won't generation any events during that time. Hence, another improved version of Queue-based model is proposed as Event-driven Queue-based model. And this model's overall performance is further increased by at least 10 times on time-step based implementation.

And since [6] is focused on performance improvements, they proposed a re-implementation of Deterministic Event-driven Queue-based simulation model, which is mentioned in the previous paper, in Java (JDEQSim), which consists of three parts: *Simulation Units, Messages,* and *Scheduler*. *Simulation Units* corresponds to vehicles and links, and they communicate via *Messages*. Since each message will carry a time stamp, the *Scheduler* will maintain all messages within a priority queue. And this model solves the problem of gaps in a queue, prevent gridlock, and allows simulation with multiple transportation modes. And they also presented a parallel version of JDEQSim and resulted in a speed-up by 1.6 times, though I didn't read into detail of this parallel version since my current focus is on the model.

III. Parallelization Techniques

As my focus is concentrate on parallelization part, I first dive into the current techniques used within multi-agent based transportation simulation.

As mentioned in chapter 25 of a book from one of the co-founders of MATSim, which basically cover all the findings related to MATSim before 2007, three parallelization techniques are proposed and used within the current implementation are domain decomposition, graph partitioning, and adaptive load balancing.

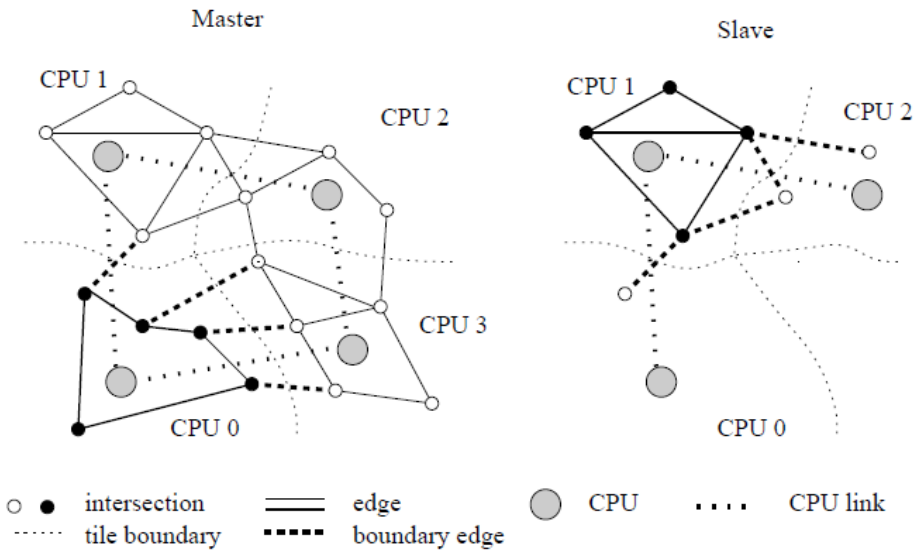

Figure 5. Domain decomposition from different perspectives [1]

Domain decomposition basically means that the simulation region is divided into different domains by their geographical location, and then each CPU will assign to deal with all the computation within one domain, as shown in Figure 5. Therefore, if we can manage to achieve similar size for all the domains, it will make the overall simulation more efficient. However, its drawback is also quite obvious. Because each CPU is only responsible of its own region, computations requiring data from other regions will somewhat suffer, hence exchange boundary and active range is discussed for solution for this drawback.

Graph partitioning is related to the part of dividing regions in domain decomposition above, and the author discuss one their original solution and METIS library. A result using orthogonal recursive bi-section of METIS library is shown in Figure 6. Based on the efficiency analysis of load on partition, the load imbalance become larger with more CPUs. Hence, the article proposed an adaptive load balancing solution, which adjust load on CPUs based on the execution time of each link and each intersection during run time.

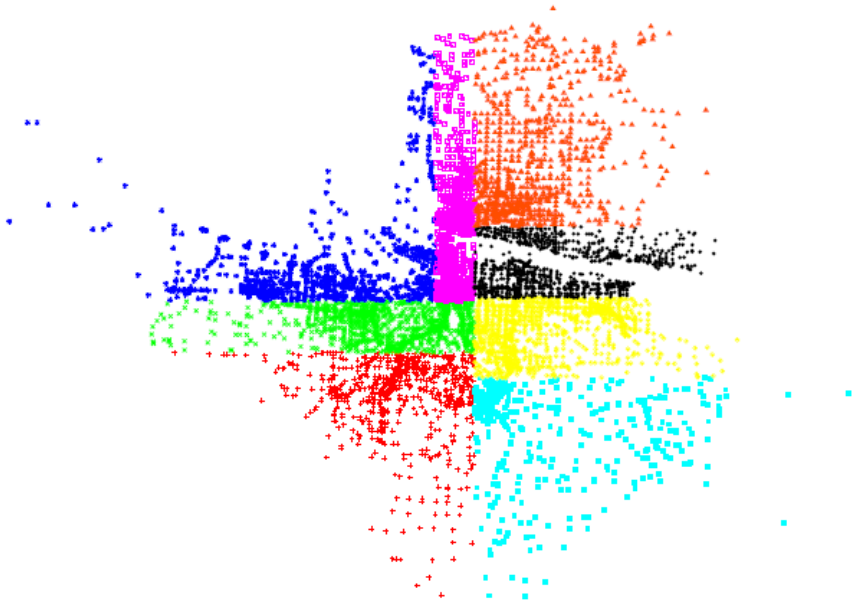

Figure 6. Graph partitioning using orthogonal recursive bi-section [7]

IV. Problem and Motivation

However, as large scale simulation, the computational requirements could become quite large, which beyond the capability of single computing node. For instance, it takes about 15 minutes on average for running each iteration under the Greater Zurich scenario with 188,000 agents, which is significantly slow compared to performance of other simulation applications. [8] Hence, the execution performance and the time of waiting for results can be unacceptable.

To improve this situation, I propose to design and develop a traffic simulation that applies distributed and parallel computing techniques. More specifically, I will be using MASS library to parallelize the execution module within MATSim Project. Further, if the result is very promising, I would like to attempt building large scale real-time traffic simulation.

Methods

I. MATSim

MATSim is a research-based framework to implement large-scale agent-based transport simulations. The framework consists of severel modules which can be combined or used stand-alone. Modules can be replaced by own implementations to test single aspects of your own work. Currently, MATSim offers a framework for demand-modeling, agent-based mobility-simulation (traffic flow simulation), re-planning, running simulations as well as methods to analyze the output generated by the modules.

As described in [9], the procedure of MATSim can be seen as an evolutionary process consists of *execution*, *scoring*, and *replanning* part, as shown in Figure 7. Therefore, after the demand and population generation, the simulation will enter an iterative loop of this process until certain condition is triggered, such as reaching the optimized demand. And the *execution* module is responsible for generating *events* for each action in the simulation, also called traffic simulation.

Figure 7. MATSim process diagram [10]

For MATSim's current implementation, I barely found papers describe it, therefore I started to look into the content in their website. As MATSim's webpage shows [10], it look very similar to the structure in [9]. Though, as I looked into its actual code, MATSim allows developers to change traffic flow simulation model as changing the configuration file and settings. After clear with general architecture and configuration, I got more clear view of what part I will going to focus working on, which are the execution and replanning, since scoring didn't require much computation.

Another dissertation in traffic demand modeling, which is based on MATSim, provide a view of process flow [11]. After reading this paper, I managed to map the traffic assignment model discussed above into MATSim structure, which corresponding to EXEC and SCORING part as shown in Figure 8 below.

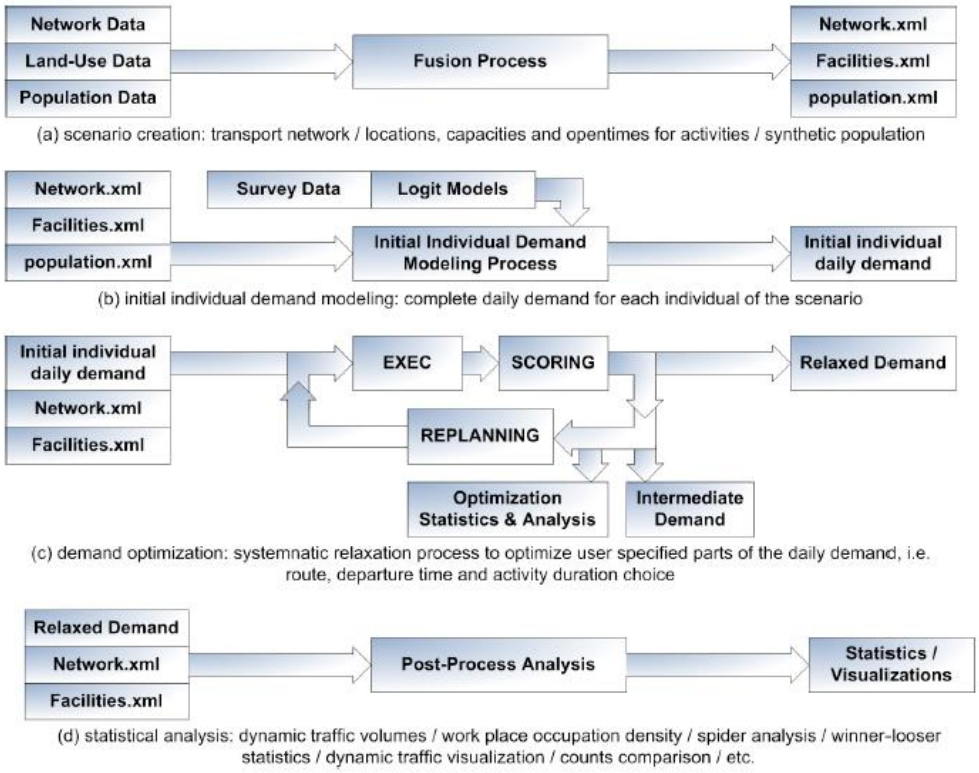

Figure 8. MATSim process structure [11]

II. Why MASS

MASS library is a parallelization library specific for multi-agent based spatial simulations, developed by the Distributed Systems Laboratory at the University of Washington Bothell. Since transportation simulation is a specific type of spatial simulation, also MATSim support multiple agents in its implementation, MASS is very suitable choice for parallelizing MATSim. For the detail regarding MASS implementation, I will introduce in the Techniques Detail section.

III. Resource and tools

Many resources were utilized throughout the execution of this project. The Distributed Systems Laboratory at the University of Washington Bothell was the primary workspace of the MASS research team in which we participated in throughout the duration of this project. The Eclipse IDE was the sole development environment for my application.

The method of source control used was EGit, combined with Maven Integration within Eclipse, and allow easy review and access to all versions of our code.

Techniques Detail

I. System Overview

Figure 9 shows the overall design of this application. The program logic begin with the central controller within MATSim, and it will utilize MASS library to distribute the computation into multiple computing nodes. After that, all nodes will only focus on its own partition of an iterative process going through *Network Reloading*, *Route Assignment*, *Traffic Simulation*, and *Parameter Exchange*.

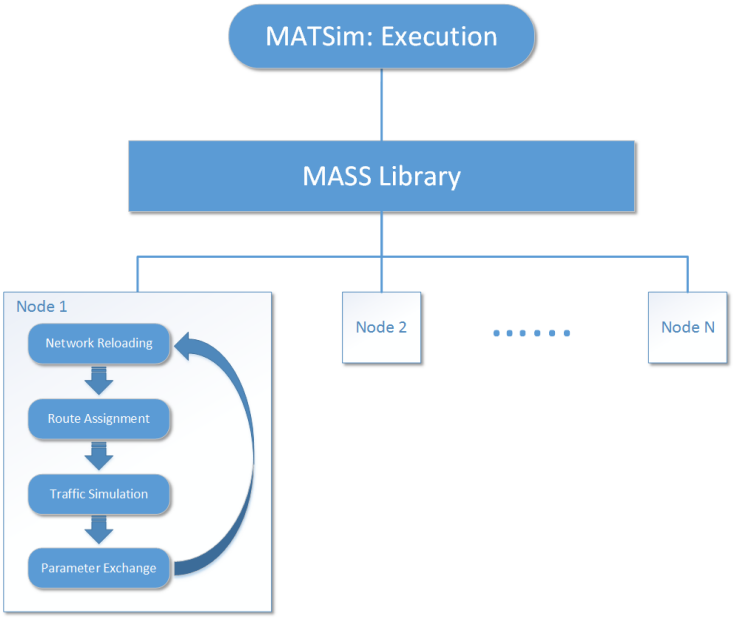

Figure 9. Overall architecture

Since the underlying implementation of MASS uses two dimensional array, all roads and intersections are stored as a single element of a distributed 2D array. However, for each element, its adjacent neighbors in the original network may not always be the elements actually stored in the array. Therefore, customized neighboring feature is added into MASS library in order to let adjacent neighbor exchange status parameters, such as buffer capacity, of their adjacent roads or intersections.

The data flow is illustrated in Figure 10 below. Each mobile agent can travel to different computing node if needed, and all computing nodes will exchange parameters with their own customized neighbors at each iteration.

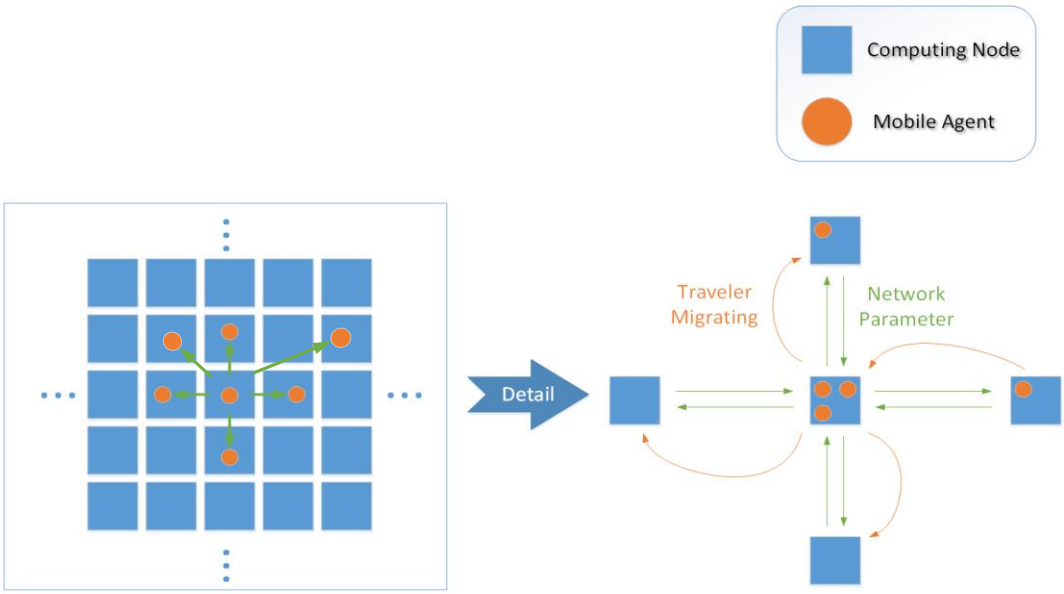

Figure 10. Data flow diagram

II. Data Structures

In terms of data structures used, adjacency list is the most important one for mapping traffic network topology into distributed two-dimensional array. Just like Figure 11 demonstrated below, the relationship of nodes and links can be converted into the structure on the right, so that it can be stored within one single list, which later will become part of each element's neighbor variable.

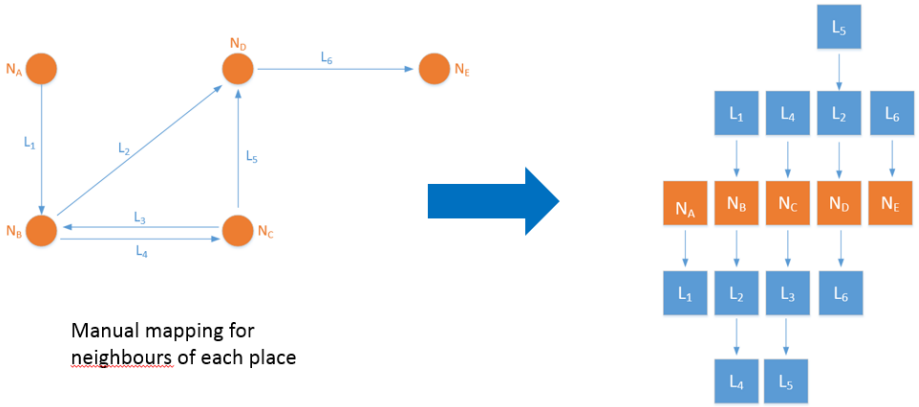

Figure 11. Network mapping using adjacent list

For the view of MASS library, all nodes and links are mapped into Place object, and the entire network is the Places object, serving as a container to hole all Place objects. And each individual traveler is supposed to be mapped into one Agent object.

III. Core Algorithms

The very core logic is located within the simulation engine, and consists of the following three steps.
Step 1: Go through all nodes, and move vehicles from node's outbuffer into their outlinks' inbuffer, as the pseudo code shown below:

```
while (going through all Node){
   Check all incoming links for buffered vehicles {
      if there is available capacity
         move vehicle to an according outlink
    }
}
```

Step 2: Go through all links, and load vehicles from their outbuffer into their central queue, as the pseudo code illustrated below:

```
while (going through all Link){
   Check buffered vehicles within the inbuffer{
      if there is available capacity
         move vehicle to the central queue
    }
}
```

Step 3: All nodes and links exchange data with their neighbours. This step basically include the following major components:

1. *Pack all network parameters into one single message*
2. *Send the message to the inMessages to all neighbor using exchangeAll*
3. *Retrive neighbor's data from each Place's inMessages*

III. Implementation Detail

As mentioned in the previous sections, network is mapped into Places object, and node and link is mapped into Place object. Originally, I design to implement two different classes according to node and link respectively. However, later I found it inefficient since it will be difficult to achieve polymorphism in MASS. In addition, I found the implementation of link and node within MATSim share quite a lot similarities, therefore, I combine these two into one Element_MASS class with a field to indicate the current object is a node or a link.

So as Figure 12 shown below, five components is highlighted in the program structure. At the bottom is the MASS component in charge of interaction with MASS library to read and exchange data. Connected

to it is the *Central Controller*, which connect to the rest three components: *Network*, *Simulation*, and *Scenario*. Within these three, simulation is the most related one, which send request and receive response from MASS component through *Central Controller*.

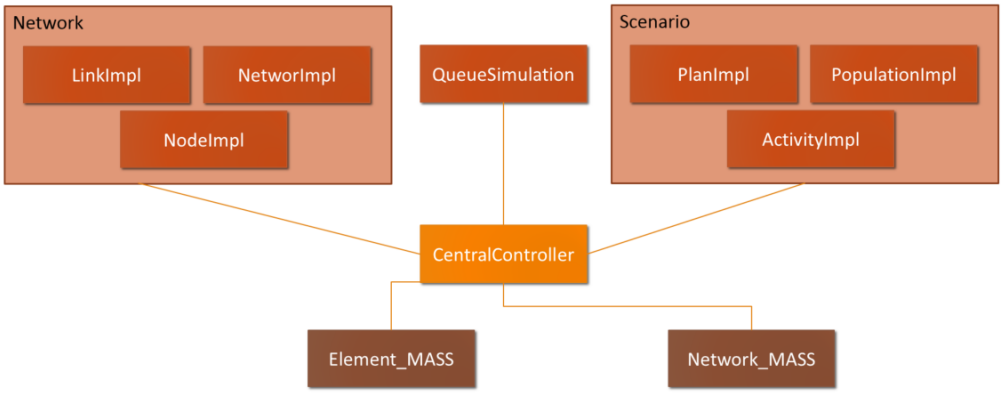

Figure 12. Program structure

A snapshot of the actual implementation of this project under Eclipse IDE is shown in Figure 13 below. On the left is my branch of the Java version of MASS library on my local machine, and I basically made the following changes to MASS library:

- Create neighbours variable within Place class to store neighbouring relationship between all place elements, along with accessor and mutator methods
- Remove destinations parameter with all exchangeAll(), and sendMessage() on EXCHANGE_ALL TYPE
- Replace destinations within Places_base with srcPlace.neighours

And the right snapshot is the project folder including files shown in Figure 12 above, as well as the Debugger class for verifying the correctness of network parameter exchange between all nodes and links.

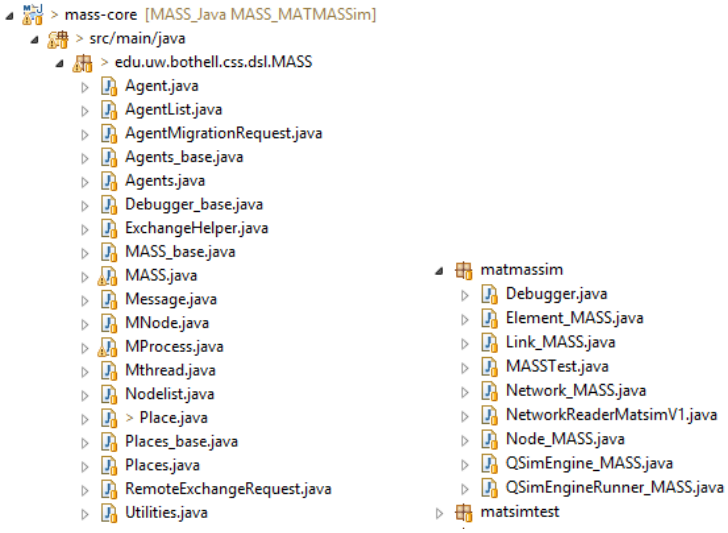

Figure 13. Project files under Eclipse

Results

I. Testing scenarios and benchmark

By far, the sample scenario within MATSim code repository was the only scenario used for collecting the current results. Within this scenario, it consisted of 4700 nodes and 13,000 links in in standard XML format like Figure 14 shown below. For each node, it only has a pair of coordinates to indicate its geographical location, and a unique id which corresponds to *from* and *to* field of a link.

```
<network name="example network">
<nodes>
    <node id="1" x="0.0" y="0.0"/>
    <node id="2" x="1000.0" y="0.0"/>
    <node id="3" x="1000.0" y="1000.0"/>
</nodes>
<links>
    <link id="1" from="1" to="2" length="3000.00" capacity="3600"
                freespeed="27.78" permlanes="2" modes="car" />
    <link id="2" from="2" to="3" length="4000.00" capacity="1800"
                freespeed="27.78" permlanes="1" modes="car" />
    <link id="3" from="3" to="2" length="4000.00" capacity="1800"
                freespeed="27.78" permlanes="1" modes="car" />
    <link id="4" from="3" to="1" length="6000.00" capacity="3600"
                freespeed="27.78" permlanes="2" modes="car" />
</links>
</network>
```

Figure 14. Input file format in XML

All results was collected on uw1-320-lab linux lab machines, each of which possess four physical CPU cores, therefore the maximum of number of thread is set to four, since it would involve context switch and affect the performance if exceed the number of physical cores.

The sequential executed version of MATSim is the benchmark for comparing my execution performance. Also within MATSim's implementation, there is a parallelized version using only Java multi-threading, therefore, the execution performance of this parallelized version has also been set as a benchmark for comparison.

II. Testing results

A. Verification

To verify the correctness of simulation process, I compare the travel distance of the sequential execution, shown in Figure 15, after each iteration from the log file generated by MATSim statistics module. As a result, they both suggest the same trend, and both starts from 9445 and end at 9429. Because it could be minor difference between each time running the simulation, also random seed is used to make decision for each traveler. Therefore, with the same trend and starting and end value, it should be considered verified.

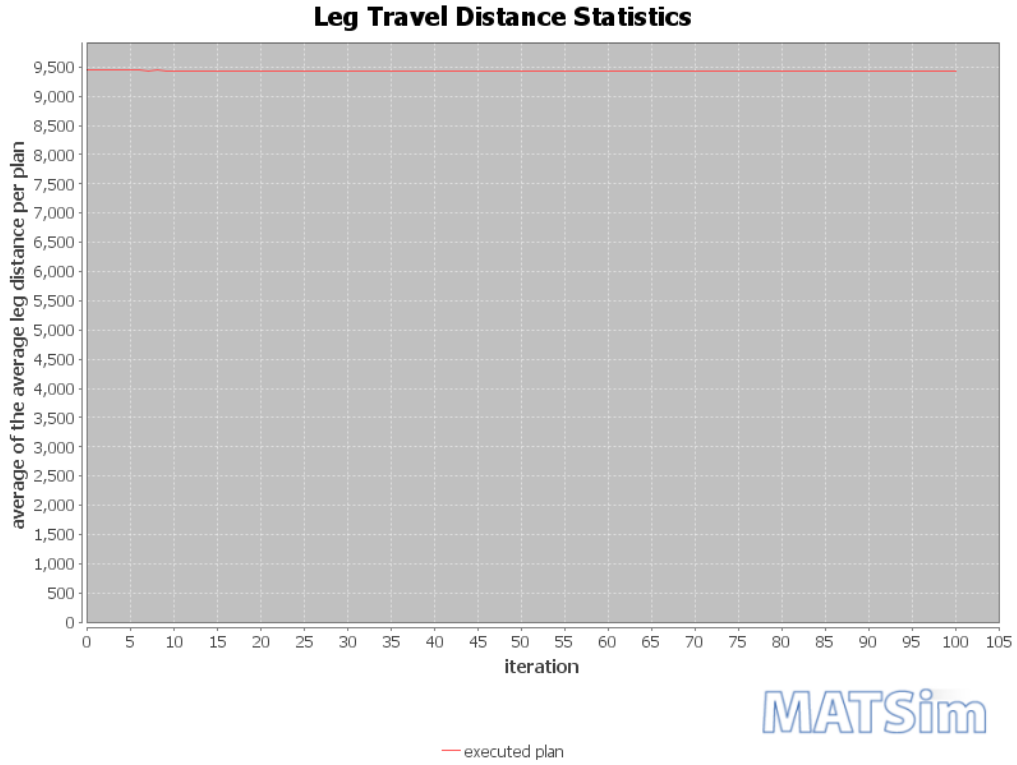

Figure 15. Travel distance output

B. Sequential Execution Performance

For the sequential execution performance, as the illustrations in Figure 16, it take about 15 minutes for completing the entire simulation, consisting of 100 iterations. Only the part in pink color is our concern, because those are the execution time for the simulation module. Also, we can ignore the increase every ten iterations, since those increase is because MATSim generated detail output every ten iterations. Therefore, only count the execution time for simulation, it took about 5 minutes to run 100 iterations, which will used as a benchmark in later comparions.

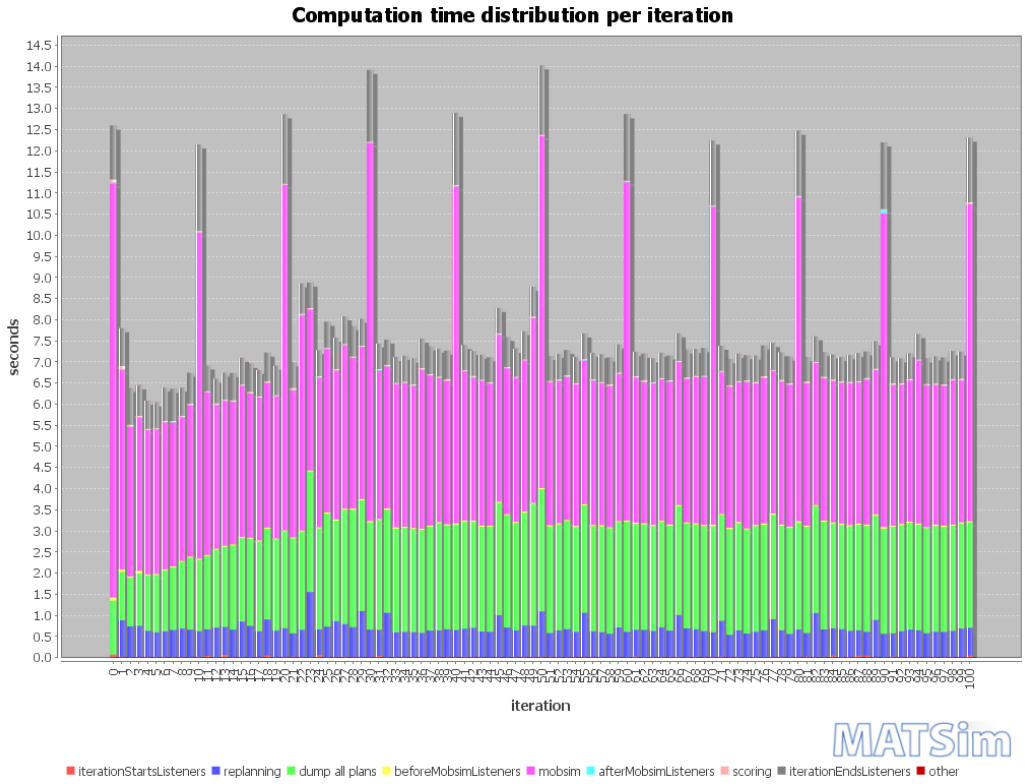

Figure 16. Sequential execution performance

C. Comparison with Sequential Execution

For the first performance comparison, results is shown with line graph showing speedup ratio start from single computing and go through 2, 4, and 8 CPUs. From Figure 17 below, we can see the performance exceed sequential execution when using more than 4 computing nodes. However, it is not quite efficient on single node or 2 nodes, but these results are all one single thread.

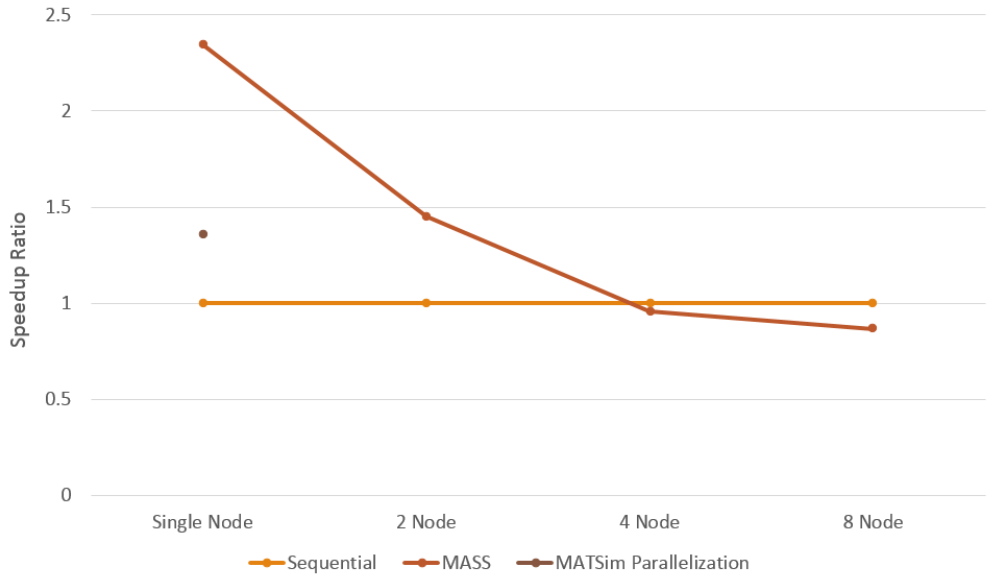

Figure 17. Execution comparison with sequential version

D. Comparison with Parallelized Version

Also I compare my performance with MATSim's parallelized version, which uses Java multi-threading, as in Figure 18 shown below, which is a speedup ratio chart showing performance of both previous parallelized version and MASS version with 1, 2, and 4 threads. Since MATSim's version only use resources of single computing nodes, and 2 and 4 nodes are used with MASS version for comparison. As the result turned out, it exceed MATSim's own parallelized version execution performance around 20%, except the point with 4 thread and 2 node using MASS.

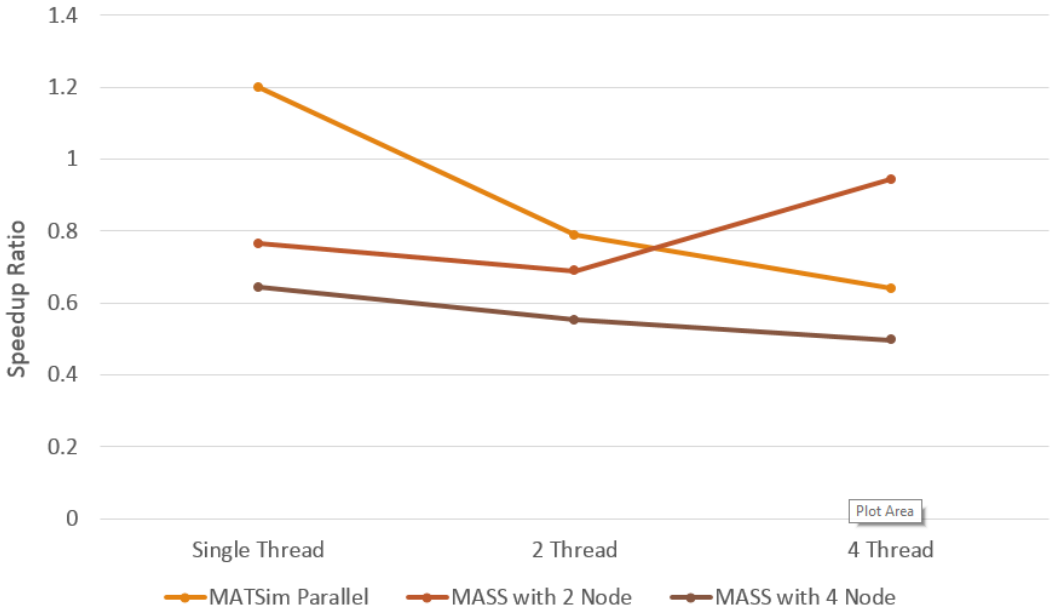

Figure 18. Execution comparison with parallelized MATSim

Limitations & Future Extension

In this section, I will discuss some of the limitation that this application have and some extension for future students can work on.

I. Partitioning Algorithm

In my implementation, because of the order of their input XML file, all the nodes is added to the distributed array first. And all links is stored to the array after, since they has to wait since it depend on the node for their *from* and *to* field. As a consequence, the partitioning is very well tuned for adjacent neighbors to exchange parameter. For most cases, all the neighbors of an element may reside on different computing nodes, so network communication has to be set up in order to exchange parameters.

Therefore, better solution need to researched and employed for map partitioning to reduce communication between nodes by deploy related Place to the same node.

I. Testing Scenarios

Because of the time and resource constraint, another two bigger datasets is not used for generating the results. Considered about the significance of the performance improvements on large data sets, the following two scenarios would be next steps for further testing, especially the Greater Zurich Area.

- Gotthard scenario
  - Have a set of 50 000 trips going to the same destination
- Greater Zurich Area
  - Consisted of 1.62 million agents, contained 163k links

Conclusion

In this paper, I present an application to parallelize the existing MATSim queue simulation module utilizing the advantage of MASS library. From the result, we can say that it has made some progress on improving the execution time, compared with both their sequential and parallelized model. But it still needs to be improved. I believe that this approach possess the potential of being the solution for better parallelization for MATSim transportation simulation tool. At the same time, a lot future work need to be done to get more improvement, as well as apply to other models within MATSim.

Acknowledgement

I would like to thank Professor Munehiro Fukuda for all the help and understanding he gave me during my capstone project development. I would also like to thank my father for the help with explaining concepts in traffic simulation. Also special thanks to Professor William Erdly, and everyone that help me in my master study in University of Washington Bothell.

Bibliography

[1] Nagel, Kai. "Multi-agent transportation simulation." Traffic 2.2 (2007).

[2] Nagel, Kai, and Gunnar Flötteröd. "Agent-based traffic assignment: going from trips to behavioral travelers." 12th International Conference on Travel Behaviour Research (IATBR), Jaipur. 2009.

[3] Peeta, Srinivas, and Athanasios K. Ziliaskopoulos. "Foundations of dynamic traffic assignment: The past, the present and the future." Networks and Spatial Economics 1.3-4 (2001): 233-265.

[4] Chiu, Yi-Chang, et al. "Dynamic traffic assignment: A primer." Transportation Research E-Circular E-C153 (2011).

[5] Charypar, David, Kay W. Axhausen, and Kai Nagel. "Event-driven queue-based traffic flow microsimulation." Transportation Research Record: Journal of the Transportation Research Board 2003.1 (2007): 35-40.

[6] Performance improvements for large scale traffic simulation in matsim. ETH, Eidgenössische Technische Hochschule Zürich, IVT, Institut für Verkehrsplanung und Transportsysteme, 2009.

[7] Cetin, Nurhan, and Kai Nagel. "Parallel queue model approach to traffic microsimulations." Proceedings of Swiss Transportation Research Conference. 2002.

[8] Balmer, M., K. Meister, and K. Nagel. *Agent-based simulation of travel demand: Structure and computational performance of MATSim-T*. ETH, Eidgenössische Technische Hochschule Zürich, IVT Institut für Verkehrsplanung und Transportsysteme, 2008.

[9] Cetin, Nurhan, et al. "Large-scale multi-agent transportation simulations."Computer Physics Communications 147.1 (2002): 559-564.

[10] MATSim Controler Strcture http://www.matsim.org/docs/controler

[11] Balmer, Michael. Travel demand modeling for multi-agent transport simulations: Algorithms and systems. Diss. ETH Zurich, 2007.

[12] Micheal Balmer. “Agent-Based Activities Planning for an Iterative Traffic Simulation of Switzerland - Activity Time Allocation” 4th Swiss Transport Research Conference (2004)

[13] Charypar, David, and Michael Balmer. A high-performance traffic flow microsimulation for large problems. Eidgenössische Technische Hochschule, Institut für Verkehrsplanung und Transportsysteme, 2008.

[14] Cetin, Nurhan, Adrian Burri, and Kai Nagel. "A large-scale agent-based traffic microsimulation based on queue model." IN PROCEEDINGS OF SWISS TRANSPORT RESEARCH CONFERENCE (STRC), MONTE VERITA, CH. 2003.

[15] Emau, John, Timothy Chuang, and Munehiro Fukuda. "A multi-process library for multi-agent and spatial simulation." Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on. IEEE, 2011.