MASS Library Traffic Simulation Application Development and Performance Evaluation

by John Spiger

Faculty Advisor: Professor Munehiro Fukuda

Table of Contents

Introduction

The MASS Library Traffic Simulation Application (MassTraffic) makes use of the MASS library (multi-threaded version) and the Multi-Agent Traffic Simulation Toolkit (MATSim). This paper focuses on the MassTraffic application, its use of the MASS library, and its integration with MATSim.

MATSim Overview

MATSim is a toolkit for running traffic simulations. The map is referred to as the **network**, intersections on the network are **nodes**, and roads between intersections are **links**. The traffic is made up of **persons**, each travelling individually. Each person is given one or more **plans** to describe a path through the network. Plans contain **activities**, which take place on links, and **legs**, which describe the path taken to get from one activity to the next.

MATSim Core

An important part of MATSim is the core. The core is essentially a set of operations executed by the Controler class (*org.matsim.core.controler.Controler*). The Controler reads in all the information needed from XML files and prepares a Config object (*org.matsim.core.config.Config*). The Config object can be used to access all the information described in the XML files.

The Controler needs three XML files, at a minimum, to get started:

1. a configuration file
2. a network file
3. a plans file

The path to the configuration file is passed in to the constructor for the Controler:

```
public Controler(final String configFileName)
```

The other two files are found in the configuration file, as shown in this XML snippet from a configuration file:

```
<module name="network">
        <param name="inputNetworkFile" value="./generated/w10_h10_p100_network.xml" />
</module>

<module name="plans">
        <param name="inputPlansFile" value="./generated/w10_h10_p100_plans.xml" />
</module>
```

The configuration file is essentially a file of parameters that the Controler uses to create the core. The network file contains lists of all the nodes and all the links in the map, and the plans file contains information about each person in the simulation and their plans.

The Core and MassTraffic

MassTraffic is designed to make use of the core as much as possible. MassTraffic must do three things to start the Controler and then the simulation:

1. Instantiate the Controler
2. Make the MassTraffic simulation known to the controller
3. Run the Controler

These three steps are done in MT_Controler in its main() method with the following commands:

```
controler = new Controler(args[0]);
controler.addMobsimFactory("MT_Sim", new MT_SimFactoryImpl());
controler.run();
```

The *controller.addMobsimFactory* call gives the Controler a factory that will be used to build the MassTraffic simulation. The "*MT_Sim*" (passed in to *addMobsimFactory* above) is the key that provides access to the MassTraffic simulation. The configuration must contain a similar "*MT_Sim*" signal to the Controler that the MassTraffic simulation is the one it should run. "*MT_Sim*" appears in the configuration file as shown in this XML snippet:

```
<module name="controler">
        <param name="outputDirectory" value="./output/test" />
        <param name="firstIteration" value="0" />
        <param name="lastIteration" value="3" />
        <param name="mobsim" value="MT_Sim" />
</module>
```

When the Controler's *run* method is called, if finds the factory for "MT_Sim" and calls the factory's one method, as specified in the MobsimFactory interface (*org.matsim.core.mobsim.framework.MobsimFactory*):

```
public Simulation createMobsim(Scenario sc, EventsManager eventsManager)
```

MassTraffic's MT_SimulationImpl implements the Simulation interface. Once the Controler has the MT_SimulationImpl, it calls the MT_SimulationImpl *run* method, which starts the simulation.

As seen above in the *createMobsim* method, a Scenario object (*org.matsim.api.core.v01.Scenario*) is given to the simulation factory. This is passed in to the simulation, also. The Scenario object provides access to the Config object metioned above.

The MATSim API

The config object, through its method calls, provides access to a host of objects implementing methods described a set of interfaces in *org.matsim.api.core.v01* and its subpackages. As an example of a subpackage, the *org.matsim.api.core.v01.network* package provides interfaces named *Node, Network*, and *Link*. These are used to handle information about the network MATSim has loaded in from the network file.

MassTraffic Overview

MassTraffic starts with the MT_SimulationImpl class. The Controler passes control to the simulation by calling the MT_SimulationImpl *run* method. The first task of the simulation is to create a network and population that can be used to run the simulation.

Starting MASS and Setting up the Network and Population

MassTraffic's network and population are implemented by the MT_NetworkImpl and MT_Population classes. These are accessed through the factory classes MT_NetworkFactoryImpl and MT_PopulationImpl, respectively.

To begin creating the network and population, the MASS environment must be started. Then the MT_NetworkFactoryImpl and MT_PopulationFactoryImpl classes are used to create instances of MT_PopulationImpl and MT_NetworkImpl. This is done in the following code of MT_SimulationImpl:

```
private void buildNetworkAndPopulation() {
        MASS.finish();// just to be sure
        String[] massArgs;
        . . . . . . .
        MASS.init(massArgs);
        MT_NetworkFactoryImpl mtnfi = MT_NetworkFactoryImpl
                        .getMTNetworkFactory(scenario);
        this.mTNetwork = mtnfi.createNetwork();
        MT_PopulationFactoryImpl mtpfi = MT_PopulationFactoryImpl
                        .getMTPopulationFactory(scenario, mTNetwork);
        this.mTPopulation = mtpfi.createPopulation();
        this.mTPopulation.reset();
}
```

Initializing the Network

MassTraffic's network is set up in the MT_NetworkFactoryImpl. First, information about the core's network is gathered by the factory's constructor:

```
private MT_NetworkFactoryImpl(Scenario sc) {
        this.scenario = sc;
        this.matsimNetwork = this.scenario.getNetwork();
        this.links = matsimNetwork.getLinks();
        this.nodes = matsimNetwork.getNodes();
        this.linksSize = links.keySet().toArray().length;
        this.nodesSize = nodes.keySet().toArray().length;
        mTNetworkImpl = null;
}
```

The call to *this.scenario.getNetwork()* is used to get the network from the core. The core network object is accessed with calls from the *org.matsim.api.core.v01.network* package.

When the factory's *createNetwork* method is called, information about the core's network is bundled up with the MT_InitLinkBundle and MT_InitNodeBundle classes. These classes are a convenience to make it easier to initialize the MassTraffic links and nodes with MASS calls.

The MassTraffic links and nodes are represented by the MT_LinkImpl and MT_NodeImpl classes, respectively. The Links and Nodes are created in the following two methods:

```
private void createMTLinks() throws Exception {
        MT_InitLinkBundle[] mTLinkBundles = getMTLinkBundles();
        mTLinks = new Places(massLinksHandle, "masstraffic.MT_LinkImpl", null,
                        linksSize);
        mTLinks.callAll(MT_LinkImpl.INIT, mTLinkBundles);
    }

    private void createMTNodes() throws Exception {
        this.mTNodes = new Places(massNodesHandle, "masstraffic.MT_NodeImpl",
                        null, nodesSize);
        MT_InitNodeBundle[] mTNodeBundles = getMTNodeBundles();
        this.mTNodes.callAll(MT_NodeImpl.INIT, mTNodeBundles);
    }
```

Once the nodes and links are created, another method, *registerMTLinksWithMTNodes*, provides the links with references to start and end nodes and nodes with references to its inbound and outbound links.

The MT_NetworkImpl object created by the factory is a singleton object.

Setting up the Population

MassTraffic's population needs the MT_NetworkImpl singleton object, so the population is set up after the network.

First MATSim's core population is accessed by the following call in MT_PopulationFactoryImpl:

```
    /* package */MT_PopulationImpl createPopulation() {
        ...
        this.matsimPopulation = scenario.getPopulation().getPersons();
```

Then the persons, MT_PersonImpl objects, are created in the same method with the following MASS call:

```
    mTPopulation = new Agents(massPersonsHandle
        , "masstraffic.MT_PersonImpl", null, mTNetwork.mTLinks,  size);
```

Eventually another MASS call is made that initializes each MT_PersonImpl with information about its starting link (i.e., starting MT_LinkImpl).

Like MT_NetworkImpl, MT_PopulationImpl is a singleton class.

Running the MassTraffic Simulation

After the MT_PopulationImpl and MT_NetworkImpl objects are ready, the simulation is ready to run.

The Timer and the Advancing of Persons

The MT_SimTimer class is used to advance the simulation time by increments. It gathers information about how to time the simulation by accessing the MATSim core:

```
    public MT_SimTimer(Scenario scenario) {
        this.startTime = scenario.getConfig().simulation().getStartTime();
        this.endTime = scenario.getConfig().simulation().getEndTime();
        this.increment = scenario.getConfig().simulation().getTimeStepSize();
```

```
                    ...
```

Each of the calls to *scenario.getConfig().simulation()* accesses information read in from this area of the XML configuration file:

```
	<module name="simulation">
		<!-- "start/endTime" of MobSim (00:00:00 == take earliest activity time/ run as long as active vehicles exist) -->
		<param name="startTime" value="00:00:00" />
		<param name="endTime" value="00:00:00" />
		<param name="timeStepSize" value = "00:30:00"/>
		<param name = "snapshotperiod"  value = "00:00:05"/> <!-- 00:00:00 means NO snapshot writing -->
		<param name = "snapshotFormat"		value = ""/> <!-- mt-viewer-live, mt-viewer-file -->
	</module>
```

The heart of the MT_SimulationImpl class is the *doSteps()* method. This method goes through a *while* loop as the timer ticks. In the *while* loop, are the following two calls:

```
	mTPopulation.advance();// each agent calculates its progress
	mTNetwork.directTraffic();// the nodes transfer the population between links
```

These lead to the following methods with MASS calls:

```
	/* package */void advance() {
		mTPopulation.callAll(MT_PersonImpl.ADVANCE);
		mTNetwork.directTraffic();
		mTPopulation.manageAll();
	}

	/* package */void directTraffic() {
		this.mTLinks.callAll(MT_LinkImpl.UPDATE_SPARE_CAPACITY);
		this.mTNodes.callAll(MT_NodeImpl.DIRECT_TRAFFIC);
	}
```

The *advance* method causes each MT_PersonImpl to recalculate its progress on the network. The *directTraffic* method causes each MT_NodeImpl to look at its incoming MT_LinkImpl links to see if any persons are waiting to move to a next link. If there is space on the next link, the MT_NodeImpl coordinates the transfer by giving a *migrate* (from the Agent class) order to persons to be transferred.

Testing

The class MT_TestFileBuilder can be used to create a test network and population. See the javadocs for information on how to do this.

Extensions

There are a variety of ways in which the MassTraffic simulation can be improved.

Viewer

The class MT_SnapshotViewer was created with the intention of providing a viewer to watch simulations. Unfortunately, there are still some problems with the display of the network and agents.

Completing MT_SnapshotViewer or creating a new type of viewer would help make the MassTraffic simulation a more robust and useful simulator.

Plan Scoring

MatSim has a way or scoring the plans of each person moving through a simulation. Unfortunately, this functionality is lost is MassTraffic. Implementing the scoring functionality would be helpful for making MassTraffic a more useful simulator.

Event Reporting

An EventsManager object is passed in to the MT_SimulationImpl constructor by the Controler. In the existing MATSim simulators, this events object is assigned to a static object that is then accessed by persons to report on events such as getting stuck or the like. Implementing the EventsManager functionality would also be a benefit to the MassTraffic simulation.

Other Types of Simulations

MATSim has four built-in simulators. They have different methods for running their simulations. Adding more types of simulators based on MASS might be useful as well.

Vehicles and Facilities

MATSim has some experimental functionality for adding vehicles and facilities to simulations. Porting this functionality over to a MASS-based simulation could very well be helpful.

Performance

A test of the MassTraffic simulation with varying numbers of threads running in the MASS environment show promising results. Running a simulation with 10,000 persons and 10,000 nodes shows improvement in running time as more threads are added:

1 thread: 50968 milliseconds

2 threads: 28617 milliseconds

4 threads: 15960 milliseconds

8 threads: 11179 milliseconds

Conclusion

The MassTraffic simulation shows promise as a useful application for the MASS library. With a multi-process version of the MASS library, large time savings might be realized in running traffic simulations.