Computational Challenges of Implementing the Atlanta Regional Commission Activity-Based Modeling System

Ben Stabler, PB

Jim Hicks, PB

Guy Rousseau, ARC

Jonathan Nicholson, PBS&J

Chris Simons, PBS&J

Joel Freedman, PB

Chuck Purvis, MTC

David Ory, MTC

November 6, 2009

Introduction

One of the key computational requirements in developing a successful Activity-Based Model (ABM) system for use at an MPO is an acceptable run time. If the run time is too long, then the model's relevance may suffer because it simply cannot provide results fast enough to keep up with the policy and planning questions being asked of it. At the Atlanta Regional Commission (ARC) and the Metropolitan Transportation Commission (MTC), an acceptable run time is an overnight run time, i.e. 16 hours. In an effort to deliver acceptable run times, the ARC and MTC ABMs were implemented with advanced software implementation strategies such as distributed/threaded processing, sampling, a shared software base, and others. In addition, the ARC and MTC model development efforts joined forces in order to take advantage of economies of scale. The purpose of this paper is to share the strategies utilized for implementing the ARC ABM.

Background

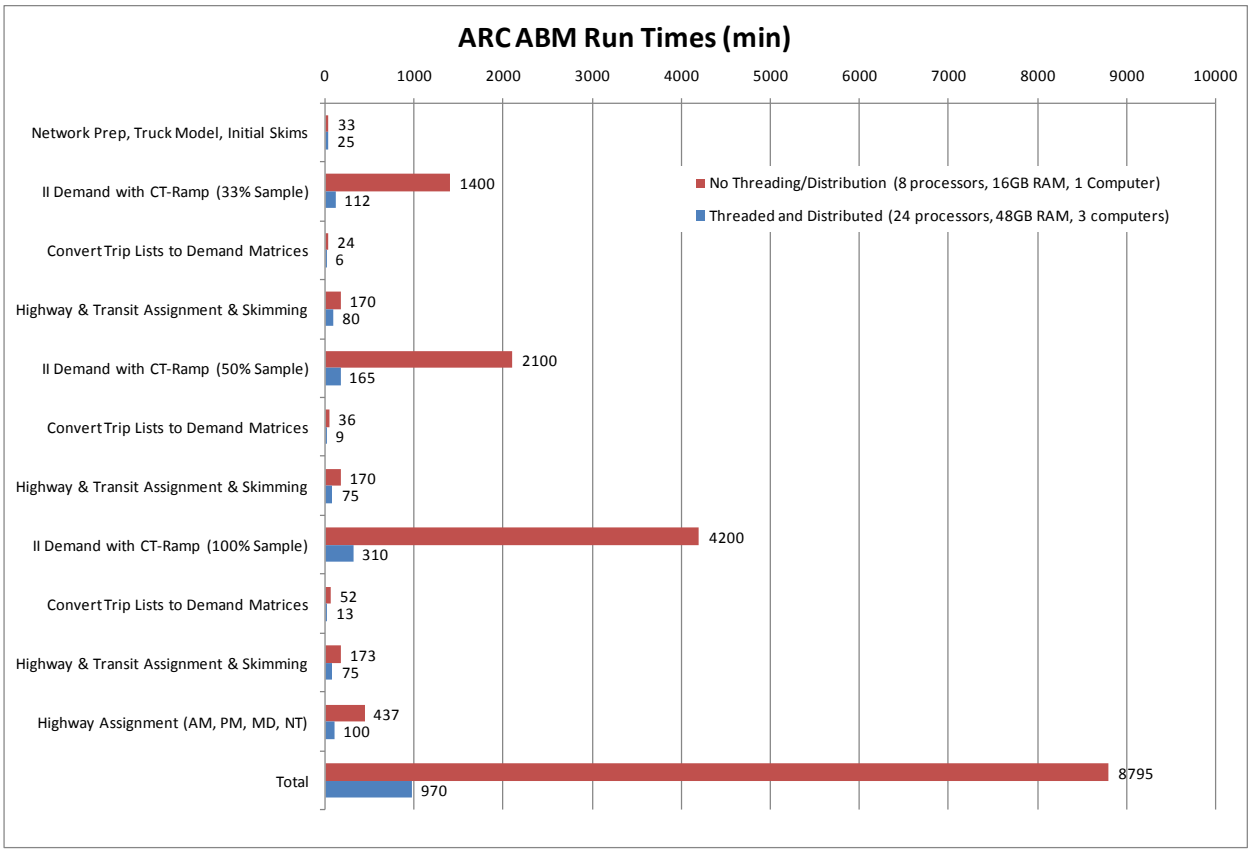
ARC decided a few years ago to invest in additional "fire-power" for its trip-based travel demand model, especially to properly address tolling and managed lanes alternatives. This led to the purchase of a dedicated computer with a dual core processor, 8 GB of

RAM, and a two seat Cube Cluster[1] license for distributed processing of Cube Scripts. This machine was used to run the trip-based model in approximately 12 hours.

Over the past several years, ARC and its consulting team was developing a new ABM based on the CT-RAMP (Coordinated Travel – Regional Activity Based Modeling Platform) family of models, which is based on models implemented in New York, Columbus, OH (MORPC), and Lake Tahoe[2]. The plan for the ARC ABM was to develop a modeling system that was distributed across multiple computers/threads and had an overnight run time.

As currently implemented, the ARC ABM runs a complete base year run in approximately 16 hours and 10 minutes. As shown in Figure 1, this includes microsimulating travel patterns for 1.76 million households, running multiple highway and transit assignments, running the entire modeling system (including network building, external demand, etc), and running three feedback loops. Without any distribution/threading, the model runs in approximately 146 hours (or approximately 6 days). The remainder of this paper describes the implementation framework developed.

Figure 1: ARC ABM Run Times



Hardware and Software

The first requirement for a distributed/threaded modeling system is a cluster of computers. ARC purchased three computers to run the ABM. The specifications of these computers are:

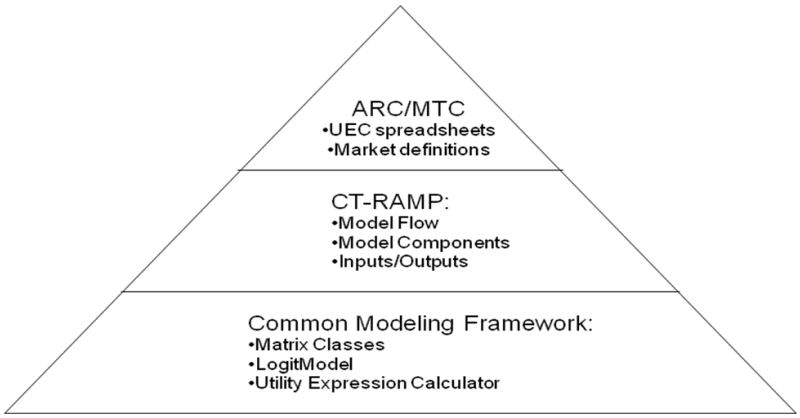
1) Windows Server 2003 64bit
2) Dual Quad Core Intel Xeon X570 2.93 GHz Processors (8 total)
3) 32 GB of RAM
4) Cube Voyager + 8 seat Cube Cluster license (16 total seats with hyper-threading)

While the hardware and general IT challenges were not insurmountable, it required some trade-offs in ARC's IT strategic plan for supporting its modeling department, with the key challenge consisting of "right-sizing" its hardware and software balance for the ABM. The total cost of the new hardware and software licenses was around $30,000.

In addition, the modeling system requires Java (32bit and 64bit versions), Cube Base for the GUI, and the CT-RAMP Java package. The model requires a 64 bit OS in order to take advantage of larger (64-bit) memory addresses. Since Cube is a 32bit application, a 32bit version of Java is required in order to natively read binary Cube matrices.

As shown in Figure 2, the CT-RAMP software for the microsimulation components of the model has been co-developed for MTC, and relies on the Common Modeling Framework (CMF), a collection of Java libraries specifically designed for the implementation of disaggregate travel demand models. Both the ARC and MTC models utilize the CT-RAMP Java package, which contains model logic, choice model structure, and model flow, while utility equations and model inputs and outputs are specific to the ARC or MTC implementation and are contained in Utility Expression Calculator (UEC) files. These Excel-based files open up the models so the parameters, input filenames, etc can be easily accessed which helps prevent errors and makes the model equations more accessible.

Figure 2: Disaggregate Travel Demand Model Software Components



Distributing and Threading the Non-CT-RAMP Components

The starting point for the ARC ABM was the ARC trip-based model. The trip-based model's internal-internal (II) demand model was replaced by CT-RAMP, while the other components were used as is or updated as needed. The other components include network processing, a commercial vehicle model, an airport model, an external model, a time-of-day model, transit network building, and highway and transit assignment and skimming. Many of these components were already threaded in the trip-based model to take advantage of ARC's dual core machine. However, additional optimization was possible now that ARC had more computing power.

The first type of threading/distribution threaded all calculations in an origin zone loop using Cube Cluster's DistributeINTRAStep command, which distributes the calculations in blocks of origin zones to waiting Cube Cluster processes. When Cube Cluster completes, it writes an end text file and the main Cube Script reads this file and continues to the next step. This type of threading is independent of the number of processes available and is flexible for adding/removing processes. Distribution by origin zone loop allowed for threading two types of calculations: highway assignments and matrix processing.

There are currently seven highway assignments; four A.M., midday, P.M., and night. Threading the assignments therefore had substantial impacts on run time. As shown in Figure 1, the distributed highway run times are approximately 4 times faster than the without distribution. The second primary process to be threaded with DistributeINTRAStep was matrix processing. This meant that much of the matrix processing, such as creating time-of-day matrices, was substantially sped up as well.

A second type of threading/distribution was done using Cube Cluster's DistributeMULTIStep command, which essentially distributes code blocks across multiple processes and then waits for all of them to complete before continuing. Unlike the first type of threading/distribution, this type requires explicitly assigning the tasks to specific processes, thereby being less flexible to adding/removing processes. This was implemented for highway assignments by time-of-day, conversion of trip lists to time-of-day matrices, and transit assignments by access mode, local/premium, and time-of-day. This type of threading improved model run times as well. For example, the amount of time to create demand matrices from the trip lists with distribution was 28 minutes compared to 112 minutes without distribution, or an improvement of almost 3 times.

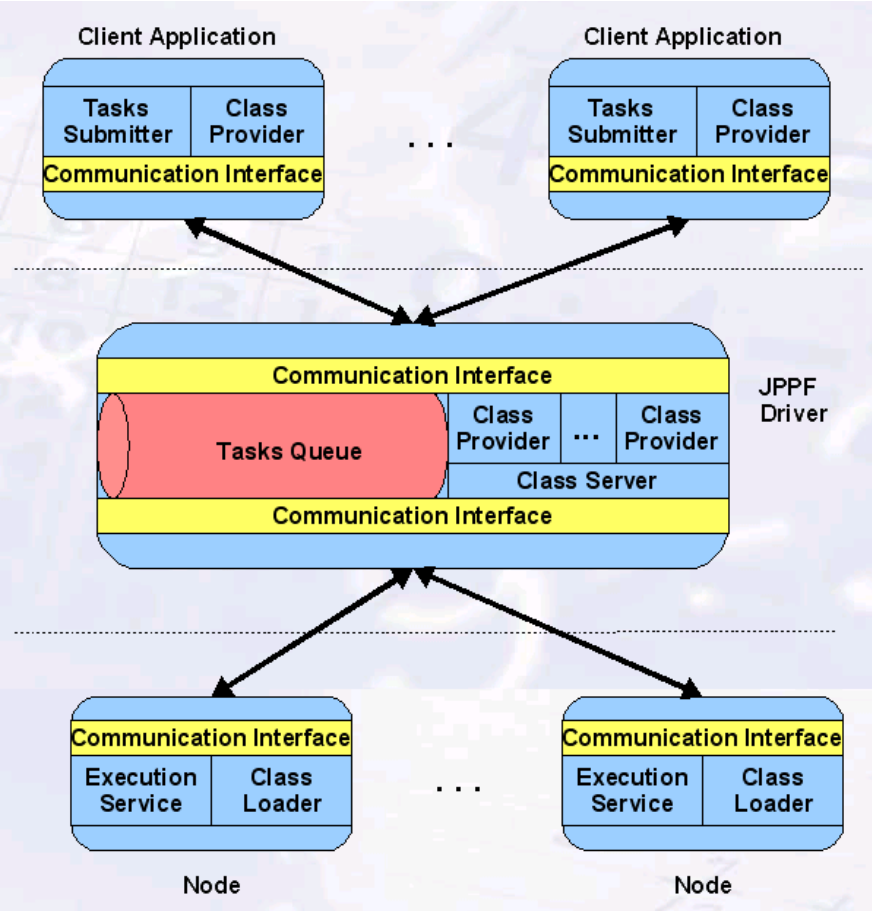
Distributing and Threading CT-RAMP

CT-RAMP, the microsimulation component of the model, is also distributed. The approach allows the utilization of one or more computers, each with multiple computing cores, and efficiently balances the computation among the computers. In addition, the configuration of the cluster of computers is relatively flexible, allowing computers to be easily added or removed. The model results, whether solved entirely on one computer or cooperatively with many computers, are identical.

Two essential design decisions were made at the start: 1) how to decompose the ABM into independent sequences of computation that could be performed in parallel, and 2) how to efficiently compute thousands of tasks in parallel regardless of the number of computers available. In CT-RAMP, activity-based choice models are applied to households and to persons independent of those applied to other households. Since households are independent, groups of households are likewise independent. Thus, the application of activity-based choice models to groups of households was distributed. Given that CT-RAMP is implemented in Java, it was decided to use a robust open source library called the Java Parallel Processing Framework[3] (JPPF) to manage the distribution of tasks. JPPF is an application written in Java that makes it easier to run applications in parallel that are written to be run in parallel.

As illustrated in the Figure below, the JPPF framework consists of 3 main parts: a driver, a set of one or more nodes, and a client. The client is in this case CT-RAMP. The nodes are also additional separate processes, typically one per computer. The driver is a separate server process that is run on one of the cluster machines. The driver is a facilitator that receives tasks from the client application, sends them to node processes, receives results from nodes, and returns those back to the client.

Figure 3: JPPF Framework



Node processes receive tasks that perform a set of calculations, perform those calculations, and return results. Nodes are configured through a properties file to communicate with the driver process upon their start-up. A typical configuration might

be to set memory equal to 32 GB and threads equal to 8 (for an 8 core machine). The majority of parallel computation in the CT-RAMP implementation occurs through tasks executed in parallel on nodes.

The driver process uses logic contained in the JPPF framework to balance computational loads across Java Virtual Machines running on the nodes in the cluster. The driver receives tasks from the client application and submits them in bundles to the nodes. The driver also retrieves class files from the client application and passes those to the nodes, as needed by the nodes. Additional nodes can therefore be added by simply editing two properties files and running a Java command.

The client application, which is called by the main Cube model script and configured through a properties file dynamically written by the model GUI, communicates with the driver as described above. The client application is responsible for creating task objects that can be run in parallel and submitting those to the driver. In the CT-RAMP implementation for ARC, 1.76 million households are split into 880 tasks of 2000 households each. The tasks are submitted to the driver and the driver assembles the 880 tasks into bundles and submits them to nodes that have notified the driver that they are part of the cluster. As the nodes complete the tasks, the driver receives their results and submits new bundles, while balancing the submission of bundles to keep the nodes uniformly busy.

In addition to the JPPF components, CT-RAMP has a Household Manager and a Matrix Manager process. The Household Manager's purpose is to manage the household and person synthesized populations in memory and to provide the JPPF nodes with all household and person related data. The Matrix Manager's purpose is to read the skims into memory and to provide the JPPF nodes with all requested skim values. These Java processes run on the main computer and have substantial memory footprints. To help reduce run time, the synthetic population is only created once in the Household Manager and then left in memory between feedback loops.

A sample of households is modeled during each feedback loop in order conserve run time. For example, for the three feedback loop run in Figure 1, the sampling shares are 33%, 50%, and 100%. The demand matrices created from the trip lists are scaled by the sampling percents before being loaded onto the network. In order to ensure that results are identical regardless of the order of processing, random number seeds are stored with each household and person so that random number cycles never go out of sequence.

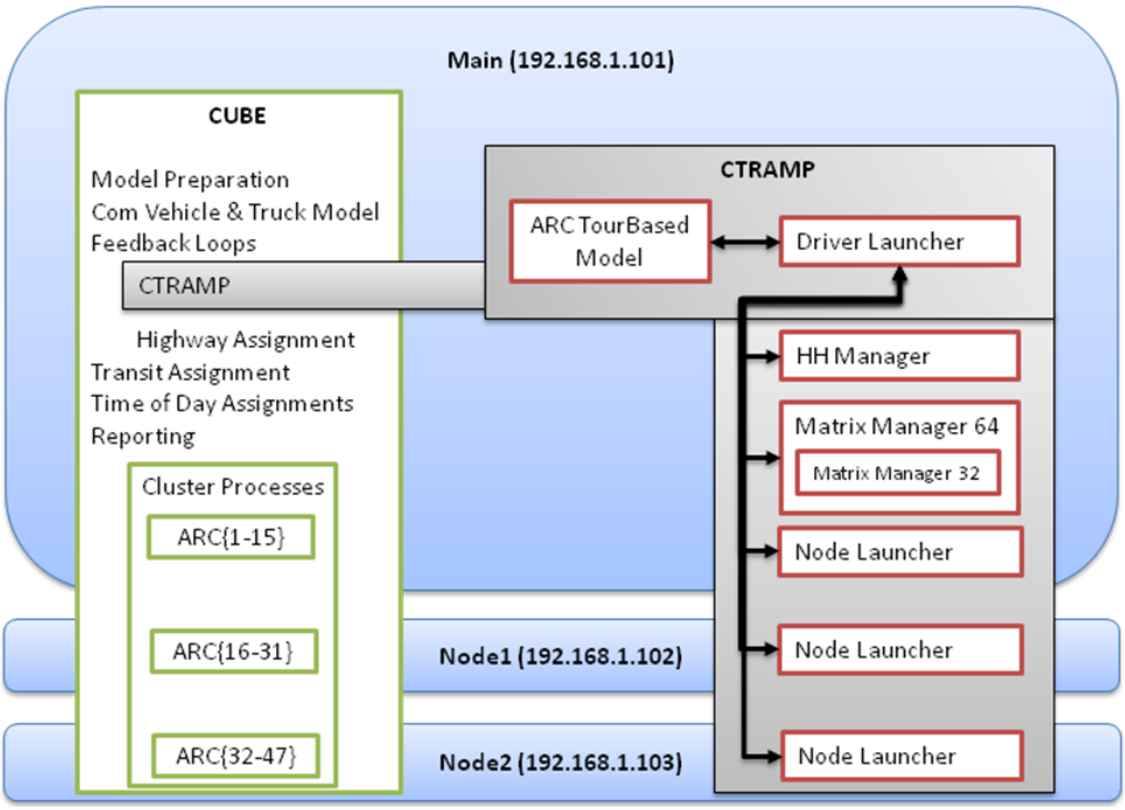
The computational gains achieved by JPPF based applications depend very much on how the tasks are programmed and to what extent they can be solved in parallel. The speedup observed with CT-RAMP was a factor of approximately 9, or 587 minutes versus 7700 minutes using the system described below.

System Setup and Design

ARC's ABM system is implemented with Java and Cube, with JPPF and Cube Cluster for distributed/threaded computing respectively. MTC's setup is slightly different, with DOS BAT files running the overall model, but the distribution/threading approach is the same. The ARC system design consists of:

1) A main computer which:
    a. runs the main Cube work, including some of the Cube Cluster processes
    b. runs the main CT-RAMP (ArcTourBasedModel) Java process
    c. runs the Household Manager Java process
    d. runs the Matrix Manager Java process (including the slave 32bit process to natively read Cube binary matrices in Java)
    e. runs the JPPF Driver Launcher processes which is called by ArcTourBasedModel, and which manages communicating with the nodes
    f. optionally runs a JPPF node process which listens for tasks from the JPPF driver
2) Two additional node computers which:
    a. Listen for tasks from the JPPF driver
    b. Listen for Cube Cluster tasks

Figure 4: ARC ABM System Design



Configuring and running the model is straight forward. The Java and Cube Cluster slave processes must be started on the two additional computers. There is a startup script for each computer that starts all required Java processes on the machine. Next the user starts the Cube Cluster processes in wait mode on each machine. The final step is to open and execute the main run script in Cube. All the model components on each machine talk to one mapped network directory. All inputs are read, and all outputs written to this folder, thereby simplifying model setup, inspection, and error detection as if the model was run on a single computer.

Conclusions and Next Steps

The current ARC trip-based model runs in 12 hours. Consistent (16 hour) run times are achieved for the ABM, allowing staff to set up a scenario before leaving for the day, and results analyzed the next morning. The substantial improvements in run times compared to an ABM without distribution/threading were made possible by a strong supply of computing power, common software, co-development with MTC, and a distributed/threaded implementation. The ability to turn around modeling results quickly is very important to the model remaining relevant to ARC's planning work.

As a new tool at ARC, the optimal configuration of the ABM remains a work in progress. Additional feedback loops, shadow pricing iterations, and increases in the synthetic population size all impact run times. Having a mapped network drive serve as the project directory makes model setup easier, but it also increases run time. And, the modeling system is built to take advantage of adding additional computers/processes to reduce run times even more. However, the law of diminishing returns applies to adding additional threads, and improvements beyond a few additional hours are not expected[4]. Longer term, ARC is interested in leveraging other computing technology, including possibly cloud computing.

References

[1] Cube Cluster, http://www.citilabs.com/cube_cluster.html

[2] Activity-Based Travel Model Specifications: Coordinated Travel – Regional Activity Based Modeling Platform (CT-RAMP) for the Atlanta Region. (2009). Parsons Brinckerhoff.

Activity-Based Travel Model Specifications: Coordinated Travel – Regional Activity Based Modeling Platform (CT-RAMP) for the San Francisco Bay Area. (2009). Parsons Brinckerhoff.

Willison, C. (2007). Development of an Activity-Based Model in the Lake Tahoe Region. 11th TRB Planning Applications Conference. http://www.trb-appcon.org/TRB-Presentations/Session%208/1_2007TRBPresentation_ChristiWillisonNew.ppt

Vovsha, P. (2005). Activity-Based Models in Practice: The Mid-Ohio Regional Planning Commission (MORPC) Model. 84th TRB Meeting, http://www.trb-forecasting.org/TRBWorkshop-158_Vovsha.pdf.

Vovsha, P. Peterson, E., and Donnelly, R. (2002), Microsimulation in Travel Demand Modeling: Lessons Learned from the New York Best Practice Model. Transportation Research Record 1805. Pages 68-77. http://tmip.fhwa.dot.gov/resources/clearinghouse/373.

[3] Java Parallel Processing Framework, http://www.jppf.org

[4] Martimo, M. Shoaib, L. Siu, V. (2007). Reducing Model Run Times Using Distributed Computing: An East-West Gateway COG Case Study. 11th TRB Planning Applications Conference. http://www.trb-appcon.org/TRB-Presentations/Session%2019/9-Shoaib%20-%20Session%2019.ppt.