AFRL-AFOSR-VA-TR-2016-0096

Dynamic Systems for Individual Tracking via Heterogeneous Information Integration and Crowd Source Distributed Simulation

Richard Fujimoto
GEORGIA INST OF TECH ATLANTA

12/04/2015
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 11-30-2015	**2. REPORT TYPE** Final Report	**3. DATES COVERED** *(From - To)* 03-14-2013 to 08-31-2015

4. TITLE AND SUBTITLE
Dynamic Systems for Individual Tracking via Heterogeneous Information Integration and Crowd Source Distributed Simulation

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-13-1-0100

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Fujimoto, Richard
Hunter, Michael
Park, Haesun

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
School of Computational Science and Engineering
School of Civil and Environmental Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR
875 North Randolph St., Rm 3112
Arlington, VA 22203-1954

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Tracking the movement of individuals in complex urban environments using mobile sensors is a challenging, but important problem in applications such as law enforcement, homeland security and defense. The Dynamic Data Driven Application Systems (DDDAS) paradigm offers a natural approach to attacking this problem. This two-year research project explored new computational technologies based on the DDDAS paradigm that could be applied to track vehicles in real time. Research accomplishments from this project include (1) the development of approaches to improve the transient response of data driven distributed simulations, (2) development of methods for on-line data driven calibration of traffic simulations, (3) development of data analytics for real-time prediction of vehicle trajectories, (4) development of algorithms for efficient execution of replicated transportation simulations, (5) analyses of data distribution methods, (6) energy analysis of synchronization algorithms for distributed simulations, and (7) development of parallel algorithms for non-negative matrix factorization for vehicle detection.

15. SUBJECT TERMS
Dynamic Data Driven Application Systems
Distributed Simulation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Richard Fujimoto
U	U	U	UU	85	**19b. TELEPHONE NUMBER** *(Include area code)* 404-894-5615

Reset

Final Project Report

Dynamic Systems for Individual Tracking via Heterogeneous Information Integration and Crowd Source Distributed Simulation

Air Force Office of Scientific Research Grant FA9550-13-1-0100

Richard Fujimoto (PI)
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

November 30, 2015

Contents

Executive Summary 3
1 DDDAS System Overview 6
1.1 System Architecture and Testbed 6
1.2 Prediction of Vehicle Locations 8
2 Transient Response of Data-Driven Distributed Simulations 10
2.1 Introduction 10
2.2 Delayed Response in the Original Ad Hoc Queueing Simulation Method 11
2.3 Iterative Ad Hoc Queueing Simulation Method 13
2.4 Experiments and Results 16
2.5 Conclusions 18
3 Online Calibration of Traffic Simulation Models 19
3.1 Introduction 19
3.2 Related Work 19
3.3 Statistical Calibration Procedure 20
3.4 Applying the Calibration Procedure 23
3.5 Discussion 25
3.6 Conclusion 28
4 Predicting Vehicle Trajectories in Real-Time 30
4.1 Introduction 30
4.2 Past and Future Trees 31
4.3 Experimental Evaluation 35
4.4 Conclusion and Future Work 38
5 Efficient Execution of Replicated Transportation Simulations 39
5.1 Introduction 39
5.2 Related Work 39
5.3 Simulation Algorithm 40
5.4 Experimental Evaluation 43
5.5 Conclusion and Future Work 45
6 Data Distribution Management for On-Line Simulations 46
6.1 Introduction 46
6.2 Approaches to Data Distribution Management 47
6.3 Power Consumption Tradeoffs 49
6.4 Scenario and Experimental Setup 50
6.5 Power Consumption: Computation 51
6.6 Power Consumption: Communications 52
6.7 Conclusions and Future Research 55
7 Energy Consumption in Distributed Simulations 57
7.1 Introduction 57
7.2 A Motivating Application 58
7.3 Related Work 58
7.4 Energy and Power Metrics 59
7.5 Distributed Simulation Systems 59
7.6 Experimental Configuration and Benchmark Application 61
7.7 Discussion 64
7.8 Conclusions and Future Work 64
8 A High-Performance Parallel Algorithm for Nonnegative Matrix Factorization 66
8.1 Introduction 66
8.2 Preliminaries 67
8.3 Related Work 68
8.4 Foundations 69
8.5 High Performance Parallel NMF 72
8.6 Experiments 75
8.7 Conclusion 78
9 References 80

Dynamic Systems for Individual Tracking via Heterogeneous Information Integration and Crowd Source Distributed Simulation

Executive Summary

Tracking the movement of individuals in complex urban environments using mobile sensors is a challenging, but important problem in applications such as law enforcement, homeland security and defense. The Dynamic Data Driven Application Systems (DDDAS) paradigm offers a natural approach to attacking this problem. This two-year research project explored new computational technologies based on the DDDAS paradigm that could be applied to track vehicles in real time. Several technical challenges were addressed in areas such as dynamic data-driven distributed simulation, analytics to exploit crowd-sourced and online sensor data, vehicle trajectory analysis and prediction, and image analysis for recognition of vehicles. New avenues of research in distributed simulation focusing on energy consumption issues were explored as part of this research in order to create effective distributed simulations executing on mobile computing platforms, an area not widely studied to date in the distributed simulation research community.

A computational architecture was developed based on the DDDAS approach to frame and address computational issues in the vehicle tracking problem. This architecture includes image processing algorithms for vehicle detection, predictive data analytics coupled with distributed simulations to assess likely future locations of the vehicle, and reconfiguration of the sensor network to maintain tracking.

This research project achieved several advances to further the development and exploitation of DDDAS technologies. Principal research accomplishments and results include:

- Transient response of data-driven distributed simulation. We envision the use of distributed simulations operating on mobile devices in close proximity to the physical system, e.g., simulations executing within a mobile sensor network that may be used to automatically reconfigure the sensor network. Methods to improve predictions of transient behaviors by data-driven distributed simulations were developed and analyzed. An important, distinguishing aspect of this research concerns the use of a distributed simulation approach that emphasizes realization within the sensor network itself to guide online decision making rather than reliance on centralized back-end computing facilities.
- Methods for on-line data driven calibration of traffic simulation. An approach to realize self-calibration of distributed simulations utilizing live data streams was developed for microscopic simulations of vehicular traffic. This approach was evaluated utilizing data collected on sections of roads in the Atlanta metropolitan area.
- Data analytics for real-time prediction of vehicle trajectories. Predictive simulations for the vehicle tracking problem require the prediction of likely future routes of the target vehicle. A data analysis approach and data structures were developed to predict likely future routes of the target vehicle utilizing historical information as well as recent location information of the vehicle in question.
- Algorithms for efficient execution of replicated transportation simulations. The proposed DDDAS approach utilizes multiple replicated simulations to assess likely future locations of the vehicle being tracked. New algorithms were developed to realize efficient replicated simulations of vehicle traffic that exploit the fact that the different runs will have much in common. Techniques to avoid repeating common computations and to eliminate computations that do not impact results needed for decision making were developed. These algorithms were evaluated for typical vehicle simulation scenarios.
- Analysis of approaches to data distribution. Approaches to efficiently distribute data from sensors to the distributed simulation were evaluated. Two approaches to data distribution based on APIs defined in the High Level Architecture standard for distributed simulation were assessed. These analyses considered the energy required to efficiently distribute sensor data for DDDAS usage scenarios.
- Energy analysis of synchronization in distributed simulation. Preliminary work was conducted to experimentally evaluate the energy consumed by synchronization algorithms for distributed simulation and to identify behaviors that significantly impact energy use. To our knowledge, these studies represent the first research to evaluate this issue, potentially opening new avenues of inquiry by the distributed simulation community. These initial studies focused on two conservative synchronization algorithms widely used in the distributed simulation field.

- Parallel algorithms using non-negative matrix factorization for vehicle detection. Exploitation of nonnegative matrix factorization algorithms for detecting vehicles from video traces was examined. An approach to achieve efficient parallel execution of the algorithms was developed and evaluated.

The remainder of this report discusses each of these research activities and accomplishments in detail. Specifically, the following chapters present (1) an overview of the software architecture used by an envisioned DDDAS system for vehicle tracking as well as a testbed implemented in the midtown area of Atlanta, Georgia in the United States, (2) the distributed simulation approach, termed ad hoc distributed simulations, and methods to accurately model system dynamics using queueing networks as a sample application, (3) methods to calibrate online traffic simulation models, (4) an approach and data structure to store historical vehicle trajectories in order to predict the forward trajectory of observed vehicles, (5) an algorithm to speed up replicated microscopic traffic simulations, (6) analysis of data distribution management approaches for the envisioned DDDAS system, (7) energy consumption of distributed simulation algorithms, and (8) fast algorithms for nonnegative matrix factorization (NMF).

Publications

Research results described in the following publications covering the above topics were support in part or entirely by this two-year contract:

1. R. M. Fujimoto, "Parallel and Distributed Simulation," *Winter Simulation Conference,* December 2015.
2. A. Tolk, C. D. Combs, R. M. Fujimoto, C. M. Macal, B. L. Nelson, P. Zimmerman, "Do We Need a National Research Agenda for Modeling and Simulation?" *Winter Simulation Conference,* December 2015.
3. R. M. Fujimoto and A. Biswas, "An Empirical Study of Energy Consumption in Distributed Simulations," *IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications,* October 2015, ***best paper award***.
4. P. Pecher, M. Hunter, R. M. Fujimoto, "Efficient Execution of Replicated Transportation Simulations with Uncertain Vehicle Trajectories," *2015 International Conference on Computational Science, Dynamic Data Driven Application Systems and Large-Scale-Big-Data and Large-Scale-Big-Computing Workshop*, June 2015.
5. R. M. Fujimoto and A. Biswas, "On Energy Consumption in Distributed Simulations," *Principles of Advanced Discrete Simulation,* June 2015.
6. T. A. Wall, M. O. Rodgers, R. M. Fujimoto, M. Hunter, "A Federated Simulation Method for Multi-Modal Transportation Systems: Combining a Discrete Event-Based Logistics Simulator and a Discrete Time Step-Based Traffic Microsimulator," *Transactions of the Society for Modeling and Simulation Intl.,* Vol. 91, No. 2, pp. 148-163, February 2015.
7. P. Pecher, M. Hunter, R. M. Fujimoto, "Past and Future Trees: Structures for Predicting Vehicle Trajectories in Real-Time," *Winter Simulation Conference*, December 2014.
8. L. Yilmaz, S. Taylor, R. M. Fujimoto, F. Darema, "Panel: The Future of Research in Modeling & Simulation," *Winter Simulation Conference*, Dec. 2014.
9. R. M. Fujimoto, A. Guin, M. Hunter, H. Park, R. Kannan, G. Kanitkar, M. Milholen, S. Neal, P. Pecher, "A Dynamic Data Driven Application System for Vehicle Tracking," *2014 International Conference on Computational Science, Dynamic Data Driven Application Systems Workshop*, June 2014.
10. S. Neal, G. Kanitkar, R. M. Fujimoto, "Power Consumption of Data Distribution Management for On-Line Simulations," *Principles of Advanced Discrete Simulation,* May 2014.
11. W. Suh, M. Hunter, R. M. Fujimoto, "Ad Hoc Distributed Simulation for Transportation System Monitoring and Near-Term Prediction," *Simulation Modeling Practice and Theory*, Vol. 41, pp. 1-14, February 2014.
12. Y.-L. Huang, R. M. Fujimoto, C. Alexopoulos, M. Hunter, "On the Transient Response of Open Queueing Networks Using Ad Hoc Distributed Simulations," *Winter Simulation Conference,* December 2013.

Awards

The paper "An Empirical Study of Energy Consumption in Distributed Simulations" co-authored by Richard Fujimoto and Aradhya Biswas won the best paper award at the IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications in October 2015. This work describes initial results evaluating the energy consumed for time management in distributed simulations in the context of creating mobile DDDAS applications, potentially opening a new area of research for the parallel and distributed simulation field.

Project Personnel

Among the principle investigators, Fujimoto led work on data-driven distributed simulation methods and the overall DDDAS approach. Hunter led work concerning transportation data collection, analysis, and simulation, and Park led the work concerning nonnegative matrix factorization. In addition, the following individuals also contributed to the project:

- Angshuman Guin (research engineer). Worked on the testbed and data collection and analysis problems.
- Ya-Lin Huang (PhD student, graduated). Worked on the distributed simulation approach and methods to improve the transient response of data-driven distributed simulations.
- Ramakrishnan Kannan (PhD student). Worked on nonnegative matrix factorization algorithms and implementation.
- SaBra Neal (PhD student). Worked on data distribution problems.
- Philip Pecher (PhD student). Worked on vehicle destination prediction and algorithms for replicated simulations.
- Gaurav Kanitkar (MS student; graduated). Worked on data distribution approaches.
- M. Milholen (MS student, graduated). Worked on data collection and storage.
- Aradhya Biswas (undergraduate student; recently joined PhD program). Worked on experimentation, data collection and analysis of energy usage of distributed simulations.

1 DDDAS System Overview

Tracking the movement of vehicles in complex urban environments using mobile sensors is a challenging, but important problem. Complex urban environments include multiple modes of transportation such as foot, bicycle, vehicle, and mass transit, among others. Individuals may utilize different modes during a single excursion. Accurately tracking and predicting the location of an individual presents many technical challenges in areas such as image analysis, identification, recognition, scene analysis, and trajectory detection and prediction. This project focused on tracking the movement of individuals traveling in vehicles.

Dynamic data-driven application systems (DDDAS) (Darema 2004) provide an adaptive approach to addressing this problem. DDDAS approaches have been widely studied and applied to various science and engineering disciplines for a myriad of purposes. Typical applications concern system monitoring such as assessing structural and material health (Cortial et al, 2007; Farhat et al., 2006), tracking wildfires (Douglas et al, 2006; Mandel et al, 2005) and hurricanes (Allen 2007). Another important class of applications concerns system optimization. For example, in an emergency situation, alternate evacuation scenarios may be modeled and evaluated in order to minimize evacuation time. The evacuation plan may need to adapt as the evacuation evolves when unforeseen events arise (Chaturvedi et al., 2006). Additional examples include path planning for unmanned aerial vehicles (Kamrani and Ayani 2007; Madey et al., 2012), parameter tuning for computer networks (Ye et al., 2008), management of semiconductor manufacturing systems (Low et al., 2005), managing natural disasters (Patra et al., 2012; Mandel et al., 2012; Brun et al., 2012), and optimizing surface transportation systems to mitigate congestion and reduce travel delays (Fujimoto et al., 2007; Suh et al., 2014). Online simulations are also used to gain insights into the physical systems that are difficult or impossible to observe, e.g., identifying an accident using cell phone data (Madey et al., 2006; Madey et al. 2007) and determining the boundary conditions of fluid-thermal systems (Knight et al, 2007).

Here, the DDDAS processing cycle is examined with respect to vehicle tracking. This cycle is depicted in Figure 1.1. It includes a continuously repeating procedure consisting of three phases:

1. *Sense*. Fixed position and mobile sensors and crowd-sourced data are used to identify and re-identify the current location of the vehicle of interest. Sensors may include fixed position cameras or mobile sensors mounted in unmanned aerial vehicles or carried by individuals. Identifying and re-identifying the vehicle given known characteristics (type, color, make and model, etc.) using cameras of unknown or changing orientation and location present significant challenges in image processing.
2. *Predict*. Based on information concerning the last known position of the vehicle and additional information such as likely travel patterns, historical travel data and dynamic information such as road congestion, a computational framework predicts future locations of the vehicle using a combination of analytic methods and distributed simulations. The computational framework constructs a probability map that indicates the likelihood the vehicle will reside at specific locations at future points in time.
3. *Adapt*. The sensor network is adapted to maximize the probability of re-identifying the vehicle. This may imply moving UAVs and other mobile sensors to new locations, or re-directing queries and investigations in crowd-sourced data to have greater focus on different geographic locations.

1.1 System Architecture and Testbed

Online data-driven distributed simulation systems rely upon a collection of sensors, simulation clients, servers, and communication infrastructure to collaboratively model an area (see Figure 1.2). The envisioned system architecture includes data collected from a variety of sensors with some local data processing capability (e.g., image recognition software) and results made available to clients (e.g., mobile simulators) and servers. In many cases the data are aggregated locally to reduce communication requirements. Video and other traffic sensors produce data such as flow rates and average vehicle speed along links in the road network. Crowd-sourced data and other traffic information must be integrated to establish the current state of the system. Composite values of key parameters are stored and time stamped in a data structure known as space-time memory (STM). The STM resides at the server and collects estimated values of state variables (e.g., traffic flow rates for individual links of the road network) and the time values for which these estimates are valid. It acts as an intermediary and integrator for communication and data requirements. Sensors provide readings directly to the STM. In addition, simulation clients may request values from the STM or other sources.

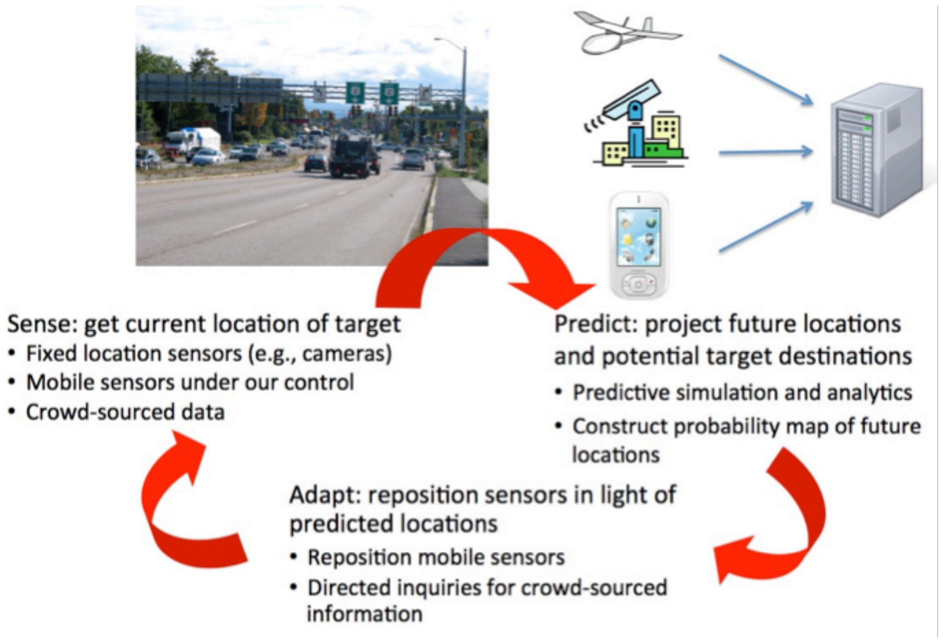

Sense: get current location of target
• Fixed location sensors (e.g., cameras)
• Mobile sensors under our control
• Crowd-sourced data
Predict: project future locations and potential target destinations
• Predictive simulation and analytics
• Construct probability map of future locations
Adapt: reposition sensors in light of predicted locations
• Reposition mobile sensors
• Directed inquiries for crowd-sourced information

Figure 1.1: DDDAS processing loop for vehicle tracking.

Software executing data analysis algorithms and predictive simulations are used to compute a probability map that estimates the likelihood the monitored vehicle will reside in particular locations at some time in the future. These computations may be performed at the server, or in some cases through a distributed computation executing in the clients. In the latter case a methodology called ad hoc distributed simulations has been developed where clients cooperate to collectively create a simulation to estimate the future state of the system (Fujimoto et al., 2007).

Communication is handled through Run-Time Infrastructure software termed the TRTI motivated by the modeling and simulation High Level Architecture (HLA). Clients may poll for information concerning specific types of data for geographic regions in order to receive all sensor and simulation messages published for that region. This also allows for mobile nodes because simulators may successively subscribe and unsubscribe to regions. Here, the Data Distribution Management (DDM) services defined in the HLA are used to provide these communication services. Because the DDM services utilize mobile clients, power consumption is an important consideration in defining suitable algorithms to implement the DDM services.

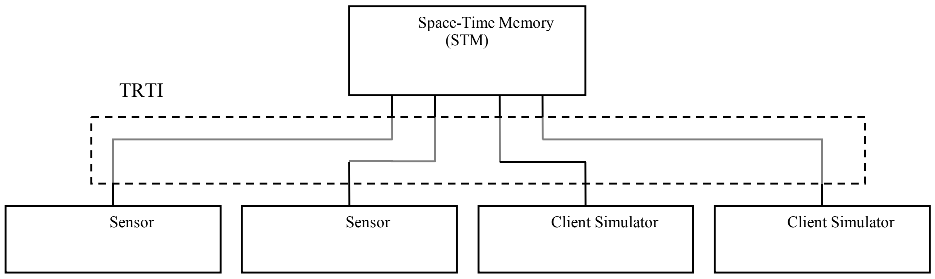


Figure 1.2: Overall system architecture. All communication is routed through the TRTI.

A testbed including cameras and communication infrastructure to monitor a portion of the midtown Atlanta region adjacent to the Georgia Tech campus creates a physical realization of this architecture. The testbed includes nine fixed location cameras that provide live video streams of vehicle and pedestrian traffic along road segments covering a four block region near the intersection of 5th and Spring street, a busy intersection surrounded by businesses and office complexes (see Figure 1.3). This testbed is augmented with mobile cameras as well as the use of LPR (License

Plate Recognition) technology to assist in vehicle identification. The testbed includes wireless LAN communication throughout the testbed area.

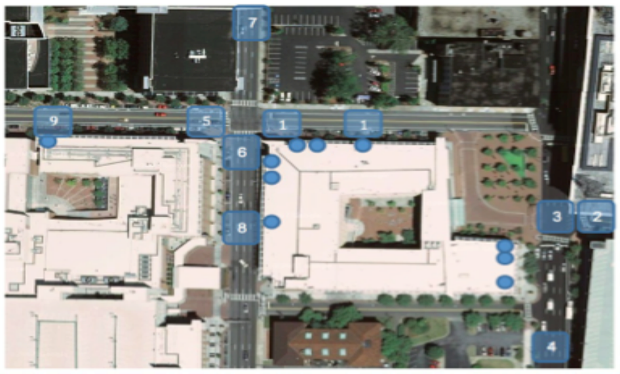


Figure 1.3: Locations of cameras and viewed areas within the testbed.

This testbed provides an infrastructure for real-world experimentation with data-driven simulations. The current implementation supports individual simulation clients reading and writing values to the STM while simulating the testbed area. In addition to client simulators, sensors and "the crowd" provide instrumented data to drive the simulations.

1.2 Prediction of Vehicle Locations

We present a framework to track a mobile target in a road network (without access to perfect information) by intermittent mobilization of sensors. The objective is to quickly predict future locations of the target while minimizing the number of required sensor observations. This framework dynamically selects an appropriate prediction model at runtime based upon what data is currently available.

Consider a directed graph for the road network *G(V, E)* with nodes representing intersections and links representing road segments. A *target entity* E_T, has been observed at vertex n_0 at time t_0. E_T travels through G on directed edges and through vertices. We assume E_T can neither be globally observed nor tracked in real time. It is assumed that E_T behaves in a non-evasive fashion. If E_T is on an edge, it is assumed, for simplicity, that E_T is also mapped to the vertex to which the edge points. ***n***=[n_0, n_1, n_2, ..., n_f] and ***a***=[a_0, a_1, a_2, ..., a_f] are used to denote the node-vector and the arc-vector of E_T's true, realized path. n(t) and a(t) return the nearest vertex that is being approached by E_T and E_T's current edge for a given time t, respectively. s(t) returns a scalar between 0 and 1 denoting E_T's position at t on a(t).

As soon as possible after E_T comes to a stop (at $t=t_f$), we wish to find E_T's exact location (i.e., $a(t_F)$ and $s(t_F)$). In order to accomplish this final goal, *sensor entities* E_1, E_2, E_3, ..., E_S are intermittently mobilized within the network, and each may test whether any one point in G contains E_T at timestamps t_1, t_2, t_3, ..., t_k.

The lag (in time units) between two consecutive rediscovery attempts is determined by ensuring that the likelihood of losing E_T at the next test timestamp does not exceed a user set tolerance probability p_{loss}. At the same time, we wish to maximize the lag within a test cycle, Δt_i, in order to minimize the likelihood that E_T becomes aware it is being tracked, and assume this likelihood increases monotonically as the candidate test interval decreases.

For a fixed p_{loss}, the critical lag Δt_i^* is determined so that the sensors may attempt to observe E_T after the sum of Δt_i^* and the previous test timestamp. This is accomplished by enumerating all paths that could potentially be reached by E_T after Δt_i^* time units. This is implemented with a microscopic, and data-driven traffic simulation that is called by a tree construction algorithm called FIND-OFFSET that is described below.

If at any test timestamp, none of the S sensors observes E_T, either E_T is lost or E_T stopped in the maximal hull that connects all vertices that can be reached after Δt_i^* time units from the time of E_T's last observed vertex. The search can occur in structured fashion, from the last observed point to the edge of the hull with S search paths that occur in parallel (along points that have the highest probability mass). In this final mode, each sensor has a fixed speed of testing a segment on an edge (length per time unit).

Once S and p_{loss} are determined, the S mobile sensors are placed at the S locations most likely to contain E_T at each rediscovery attempt, the third phase of the DDDAS loop described earlier. Therefore, the key is to construct an accurate conditional probability map of E_T's location in G at time t_{i+1} given the last known positions of E_T at time t_i, t_{i-1}, ..., t_0. Several methods are presented to estimate ($a(t_{i+1})$, $s(t_{i+1})$).

To summarize, the high-level tracking framework algorithm follows:

```
1 Fix S and $p_{loss}$
2 While( $E_T$ was detected in the last test )
3   Determine the critical lag, $\Delta t_i$* (FIND-OFFSET)
4   Determine the probability map @$t_i+\Delta t_i$* (PROBABILITY-MAP)
5   Place the $E_1$, $E_2$, $E_3$, ..., $E_S$ at the S most likely locations
6   Test at $t_i+ \Delta t_i$*
7 Find $E_T$'s stopping point within the reachable hull
```

***FIND-OFFSET: Determining the critical lag*.**

The objective of FIND-OFFSET is to find the critical lag to the next test, the maximum time that E_T can go unobserved, while still satisfying p_{loss}. This is done with a tree construction algorithm, where the vertices represent intersections and arcs represent roads between those intersections. The algorithm first starts in a global search (roughly, level-by-level) and then proceeds to engage in a local search with an S-element window (node-by-node) until the critical lag is identified. If p_{loss} is 0, the algorithm can be sped up substantially by simply checking for cardinality and side-stepping the call to the evaluation function PROBABILITY-MAP (which is not redundant if p_{loss} is 0 because the final test mode also invokes it).

PROBABILITY-MAP: Path speculation.

In order to properly assess the likelihood of different locations, all relevant and available data must be considered. In accord with the DDDAS paradigm, the system will automatically switch prediction modes, or 'tiers,' based on the data that becomes available during runtime (crowd-sourced or collected from other sources). There are three prediction tiers that this approach uses, with higher tiers utilizing more specific data, but with more promising accuracy.

- Tier 1: No or Static Data. If no historical data is available, or only general information (popularity or population density of different zones), inferences can still be made from E_T's trajectory that has been observed thus far. One particular clue to consider is efficiency. Krumm developed a static prediction model (Krumm 2006) that assigns higher likelihoods to areas that are consistent with the path taken thus far. Thus, inefficient potential trajectories are given less weight. For use in the overall procedure, one additional factor can be conditioned against – time. Since we know when the next test will occur, all cells except the ones that could be reached after the critical lag can be excluded.
- Tier 2: Past Trajectories of the Population. If E_T is currently in an area where trajectories have been observed in the past, population path similarity information can be used. One way to store this information is via decision trees holding relative frequencies at each branching point (this is discussed in detail below).
- Tier 3: Past Trajectories of E_T. The ideal situation is to have historical data available from the vehicle being tracked. Laasonen (2005) developed a model where similar routes of an individual are clustered. If an observed path is similar to an existing composite path, the composite path is merged with the observed path. Merging, in this case, refers to finding an alignment between two routes (via the dynamic programming procedure to the edit distance problem) and updating the position of intermediate locations with a new average. A similarity index is used for inserting into the appropriate cluster when a new path is observed and when the future path is to be predicted from a current location. Instead of using a quadratic time edit distance algorithm, Laasonen suggests using a heuristic, *inclusion similarity*:

$$I(r, t) = T/|t|$$

where r refers to the composite path, t refers to the path whose similarity is being computed, and T refers to the number of elements in t that appear in r in the same order. Laasonen further makes use of factors such as the time of day, the particular weekday, relative frequency, and the previously observed *base* (a base refers to a place that is deemed important to a person and is crystallized when the person spends a significant amount of time at such a place).

The route predicted by this framework is utilized by distributed simulations that predict the timeline for future movements of the vehicle taking into consideration congestion and other factors. This approach, termed ad hoc distributed simulation, is described next.

2 Transient Response of Data-Driven Distributed Simulations

Ad hoc distributed simulation is a methodology for embedded online simulation, has been studied for the steady-state simulation of transportation systems, and more generally, open queueing networks. However, for most online simulation applications, the capability of a simulation approach to respond to system dynamics is at least as important as the performance in steady-state analysis. Hence, this chapter focuses on the prediction accuracy of the ad hoc approach in open queueing networks with short-term system-state transients. We empirically demonstrate that, with slight modification to the prior ad hoc approach for steady-state studies, system dynamics can be modeled appropriately. Furthermore, a potential livelock issue that arises with the modification is addressed.

2.1 Introduction

This chapter explores the ability of ad hoc distributed simulation to predict the transient behavior of physical systems. Ad hoc distributed simulation (Fujimoto et al. 2007) is an approach to embedding online simulations into a network of sensors that monitors the system under investigation (SUI). An online simulation is a predictive computational model that utilizes the data pertaining to the current state of a SUI to project future system states. This usage of real-world, up-to-date data allows model adaptation, which in turn potentially improves prediction accuracy; this facilitates system monitoring and control.

Online simulation is also referred to as dynamic data-driven application systems (Darema 2004), symbiotic simulation (Fujimoto et al. 2002), and cyber-physical systems (Lee 2008) in the literature. These approaches have been widely studied and applied to various science and engineering disciplines for a myriad of purposes (Davis 1998). One typical application concerns system monitoring, such as examining the structural and material health (Cortial et al. 2007, Farhat et al. 2006), and tracking wildfires (Douglas et al. 2006, Mandel et al. 2005) and hurricanes (Allen 2007). Another popular application is to optimize the operations of a physical system. For example, in an emergency situation, alternate evacuation scenarios may be modeled and evaluated in order to minimize evacuation time. The evacuation plan may need to adapt as the evacuation evolves when unforeseen events arise (Chaturvedi et al. 2006). Additional examples include planning paths for unmanned aerial vehicles (Kamrani and Ayani 2007), tuning parameters for computer networks (Ye et al. 2008), managing semiconductor manufacturing systems (Low et al. 2005), and optimizing surface transportation systems (Hunter et al. 2009a, 2009b).

Capturing system dynamics is crucial to online simulations, e.g., to trigger modifications to the configuration of a physical system in response to events. For example, a sudden increase in traffic volume may indicate that the changes in traffic signal plans for the responsive transportation system are necessary to help reduce congestion. This work adopts the queueing model as the benchmarking application because the queueing model is well known for its generality and flexibility to model real-world operational systems. Examples include vehicular/air traffic, computer systems, communication networks, supply chains, and many others that involve distributing limited resources/services among users. Since most of these are time-varying systems, transient-state analysis is vital to the success of corresponding applications.

Transient-state analysis can be complex and computationally expensive, especially in fulfilling real-time requirements. Transient-state analysis concerns the system state varying over the span of time; it takes into account not only the system state at every time point but also the correlations among the prior and posterior system states. Instead, to evaluate the ad hoc approach, this study simplifies the analysis: we consider a sufficient number of time points and compare the respective state predictions from the ad hoc approach against those from the corresponding sequential simulations. Specifically, the evaluation discretizes simulation time into small time intervals, and the output measures of interest (e.g., queue-length estimates) are calculated by averaging the numbers within each interval.

The rest of this chapter is organized as follows. First, in Section 2.2, we examine the effectiveness of the preliminary ad hoc queueing simulation method introduced by Huang et al. (2012) in a scenario with increases in external arrival rates; the method reveals a delayed response in capturing the propagation of the expanded number of arrivals across modeled queueing networks. To resolve this delayed-response issue, Section 2.3 proposes a method and discusses the possible livelock issue following the new design. The new method, termed "iterative ad hoc queueing simulation method," is evaluated empirically in Section 2.4 under several network configurations, including one with a large increase in arrivals over a short period of time (which leads to rapid increases in queue occupancy). Finally, Section 2.5 concludes this study.

2.2 Delayed Response in the Original Ad Hoc Queueing Simulation Method

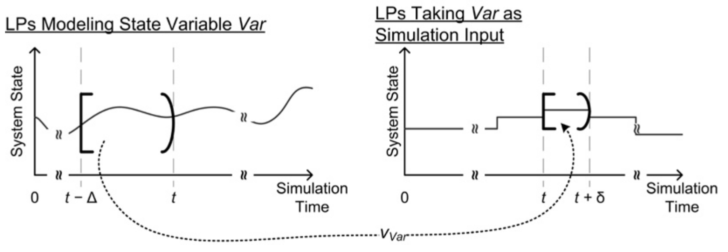


Figure 2.1: Information Sharing Mechanism Leading to Delayed Response

The existing ad hoc queueing simulation method introduced by Huang et al. (2012) is prone to delayed response to system dynamics because logical processes (LPs) share locally-observed current state information as predictions of the future. Specifically, consider the case where an LP models an object and shares the state information with other LPs that use the information as simulation input. As shown in Figure 2.1, at simulation time t, the LP computes a value based on the behavior of the object over the time period $[t-\Delta, t)$. The value becomes public as a predicted value of the object with respect to $[t, t+\delta)$, rather than $[t-\Delta, t)$. As a consequence, observations are not immediately reflected to the simulation model requiring the information, which results in delayed response.

The extent of this delayed-response issue depends on δ and Δ. The value δ is determined by the frequency in which LPs publish state information; smaller δ values allow changes to be revealed more frequently but introduce greater communication overhead in sharing information. On the other hand, the value Δ indicates the time interval needed to collect state information. While a large Δ may be used to reduce the variability of computed values, this may hide important system dynamics as the significance of system changes is mitigated by the last Δ-long history.

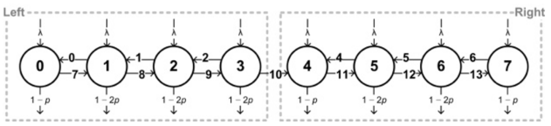
The following experiments illustrate the effects of δ and Δ. Three configurations are evaluated, all with $\delta = \Delta =$ 60, 300, and 600 seconds. The rest of the simulation model is the same as that in the previous work, and the fundamental mechanisms are as follows. Each LP models an arbitrary queueing subnetwork using a sequential, discrete-event simulation. Every Δ seconds, LPs update the mean interdeparture times on the links they simulate, and request (or estimate if the data is unavailable upon request) the same information on the input links entering their individual modeled subnetworks. The arrivals on those input links are modeled as Poisson processes, and the rates at the beginning of simulations are all set to λ (same as the external arrival rate to every queueing station); thereafter, the rates are dynamically estimated using the data from rollbacks. The rollback criteria are based on acceptable ranges, constructed by a quality control paradigm. Since there may be several predictions from the LPs modeling the same link, the data retuned to the requesting LPs are generated using a kernel density estimation (KDE) approach.

The experiments involve two open queueing networks in Figure 2.2 with the intention to show the various degrees of impact due to the delayed-response issue. The network in Figure 2.2(a) is an 8-node, partially-bidirectional tandem network. Each node represents a single-server queueing station with an unlimited buffer, FIFO/FCFS service discipline, and independent and identically distributed (IID) exponential service times with the mean equal to 1 second. Each node has external Poisson arrivals with the rate λ. The probability of a processed unit moving to another node is p. Hence a processed unit leaves the network with probability $1-p$ (at nodes 0, 4, or 7) or $1-2p$ (otherwise). The network in Figure 2.2(b) (referred to as a "completely-bidirectional tandem network") is almost the same to the former one with an exception that the processed units at node 4 may leave for node 3 with probability p.

The experiments on both the networks deploy 20 LPs in each replicate run: ten LPs modeling the leftmost 4 nodes and the remaining ten for the rightmost 4 nodes. In simulating the partially-bidirectional tandem network, shared information always goes from the left subnetwork to the right one. The right subnetwork "learns" the system dynamics in the left one through the changes in the flow rate (or, equivalently, the mean interdeparture time) of link 10. It is anticipated that the larger the Δ, the later the dynamics are detected. This phenomenon is expected to be worse for the completely-bidirectional tandem network since both the left and right subnetworks require in-formation from each other.

The evaluation of δ and Δ considers two metrics over 10 IID ad hoc runs: the arrival rate across link 10 and the mean queue length at node 4. In one run, since the 10 LPs modeling node 4 (where link 10 enters) may use different arrival rates with respect to the same simulation time, these rates are averaged into one value. Then, the mean of the 10 values (each from one run) represents the point estimate. A similar calculation is performed to estimate the mean queue length at node 4 every 60 seconds.

The results from the corresponding sequential simulations serve as the ground truth. These results are based on 100 replicate sequential runs because 10 IID ad hoc runs deploy a total of 100 LPs to model one node. Note that these sequential simulations do not model the arrivals on link 10 as Poisson arrivals (as is done by ad hoc simulations). Instead, the arrivals are the departures from node 3 filtered by the probability p. For the output, the rate is estimated by the corresponding mean interarrival time, which is computed every 60 seconds based on the arrivals appearing since the last computation.


(a) Partially-Bidirectional Tandem Network

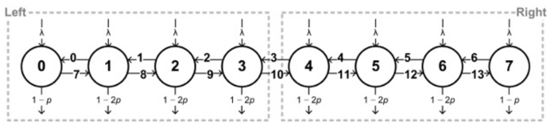

(b) Completely-Bidirectional Tandem Network

Figure 2.2: 8-Node Tandem Queueing Networks

Case 1: Partially-Bidirectional Tandem Network

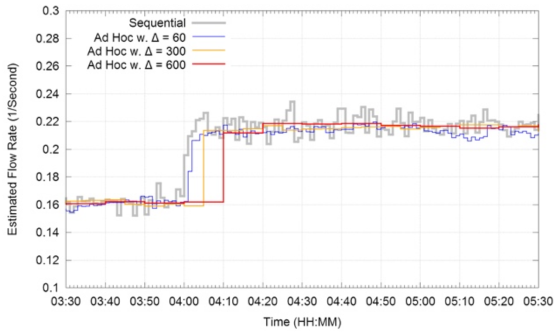
First, we study the partially-bidirectional tandem network with p = 0.45 and, for the first 4 hours in simulation time, $\lambda = 1/8$ per second; the steady-state traffic intensities of nodes range from 0.36 (nodes 1 and 4) to 0.66 (node 6). Afterwards, the external arrival rate λ increases to $1/6$ per second, causing the growth of the steady-state traffic intensities to the range between 0.48 (nodes 1 and 4) and 0.87 (node 6).

Figure 2.3 plots the experimental results from four different simulations: sequential simulations and ad hoc queueing simulations with $\delta = \Delta$ = 60, 300, and 600 seconds. In order to focus on the transient period, the prior and later parts are removed for now. On the left is the estimated arrival rate across link 10. The results match the expectation that a larger Δ value gives rise to longer delay in incorporating state changes. The delay is approximately Δ in length except in the case of Δ = 60 where the delay is slightly larger. This is because the predictions have a higher variation as they are based on a smaller amount of data. On the other hand, the right figure shows the estimated mean queue length at node 4 in the same simulation setting. Although the queue length is partly influenced by the arrivals on link 10, the discrepancy between the ad hoc runs and sequential runs is noticeable, albeit less severe.

Case 2: Completely-Bidirectional Tandem Network

This case concerns the completely-bidirectional tandem network with p = 0.4. Same as the above case, the external arrival rate λ increases from $1/8$ to $1/6$ per second after 4 hours in simulation time. Before the transition, the steady-state traffic intensities range from 0.31 (nodes 1 and 7) to 0.57 (nodes 4 and 5); following the rate change, they are between 0.41 (nodes 1 and 7) and 0.76 (nodes 4 and 5).

The experimental results are in Figure 2.4. Compared to those in Case 1, the ad hoc runs in this case take longer to pick up the state change. For example, the ad hoc runs with Δ = 600 do not fully reach the expected state until approximately one hour after the change has occurred. This prolonged delay results from the "mutual dependence" of the left and right subnetworks. In other words, the projected arrival rate across link 3 relies on that along link 10, and vice versa.



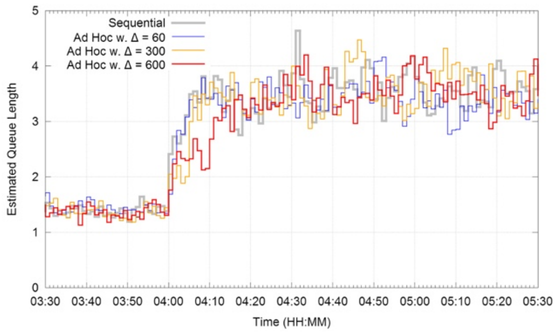


Figure 2.3: Estimated Arrival Rate across Link 10 and Mean Queue Length at Node 4 under Case 1

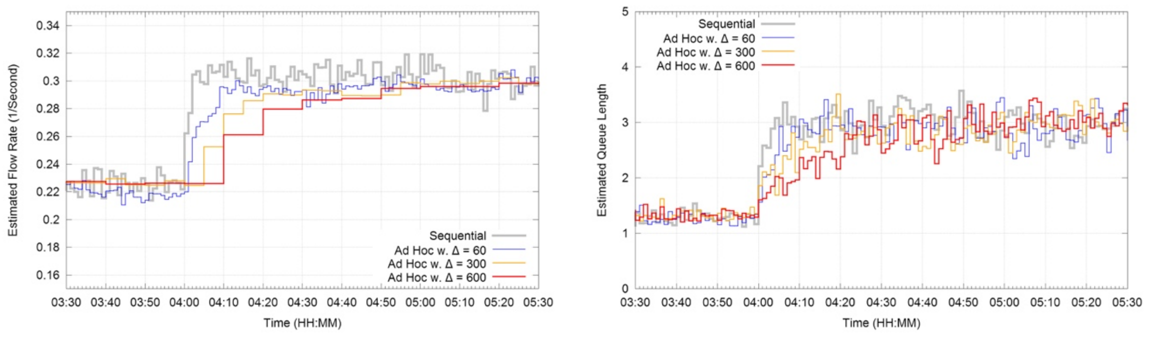


Figure 2.4: Estimated Arrival Rate across Link 10 and Mean Queue Length at Node 4 under Case 2

2.3 Iterative Ad Hoc Queueing Simulation Method

This section proposes a solution to the delayed-response problem by reassigning a new meaning to the values computed during simulation executions. These values are considered as representations of the corresponding current system states, rather than future-state predictions as in the original ad hoc method. The details are described in the following subsections, followed by a discussion of the potential livelock issue.

The proposed iterative ad hoc queueing simulation method inherits most components from the original one, including the partitioning, the local simulation models, the information aggregation, and the rollback-based optimistic synchronization mechanism. These common parts are briefly summarized to provide a comprehensive view of the method without the focus deviating from the new design.

Partitioning and Local Simulation Model

As in the original ad hoc method, a queueing network of interest is partitioned into subnetworks of various sizes and shapes with the possibility that these subnetworks may overlap with each other. Every LP models one subnetwork and uses the discrete-event simulation technique to construct its local simulation model. The simulation input is the state information of incoming links to the corresponding modeled subnetwork, and the output is that of all modeled links. The specifics regarding the link state information will soon be revisited in the next subsection.

Information Sharing

Similar to the original ad hoc method, the shared link states are represented in a high-level abstraction instead of using the exact time points of job arrivals/departures. Specifically, LPs exchange estimated flow rates every Δ seconds where an estimated flow rate of a link is computed by reversing the mean interarrival time over the last Δ seconds on that particular link. For example, let $t_i \leq t_{i+1} \leq \ldots \leq t_{j-1}$ denote the arrival times within a time interval of interest. Then, the estimated flow rate r is the reciprocal of the mean interarrival time, i.e., $r = (j - i) / (t_{j-1} - t_{i-1})$. However, generating individual arrival times out of r is not straightforward because the information of the respective interarrival-time distribution is not maintained, nor is the correlation relationship among those arrivals. Here, it is assumed that the arrivals on one link form a Poisson process with the rate equal to the corresponding estimated flow rate.

Unlike the original ad hoc approach, this iterative ad hoc method requires LPs to share link information in a way that state changes are reflected to others as soon as possible. This is accomplished by imposing that an estimated flow rate over a given time period must be used to reconstruct the link during that particular time interval. Specifically, consider a value v as the flow rate over $[t, t + \Delta)$ on some link l. The LPs modeling link l as an incoming link have to use v (along with the assumption about the corresponding arrival process) to generate arrivals during $[t, t + \Delta)$.

This design guarantees that simulations do not progress backwards since the value over $[t, t + \Delta)$ from one LP affects, at earliest, the simulations of $[t, t + \Delta)$ carried out by other LPs. Or, in the terminology of distributed simulations, the "lookahead" value is zero. Zero lookahead commonly raises the concern of deadlocks, which can be eliminated by optimistic synchronization. However, livelocks are possible because LPs may fall into a loop within which they keep rolling back each other. Avoiding such livelocks requires careful design of local simulation models; this will be revisited later.

Information Aggregation

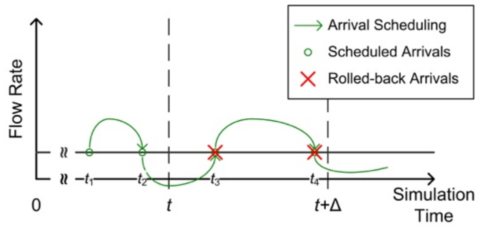
This iterative ad hoc method retains the original information aggregation mechanism—each state variable (i.e., the estimated flow rate of one link over a certain time interval) is affiliated with a data model so that aggregating various estimates is equivalent to randomly generating a sample out of the model. Data models are constructed using the KDE method with Gaussian kernels.

Optimistic Synchronization and Rollbacks

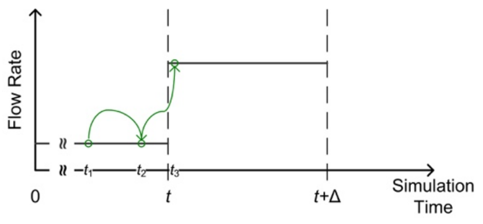
The optimistic synchronization in this iterative ad hoc method builds upon three principles: 1) intuitive practices, 2) simple implementation/maintenance, and 3) statistical validity. While the last one is the fundamental idea to determine the necessity of a rollback (referred to as the "rollback criterion"), the former two are pervasive throughout the design. For example, when a desired flow rate is unavailable, the requesting LP uses the most current rate of the same link rather than an arbitrary value, by assuming that the link state has not changed. Another example concerns the system state restoration for nonstationary Poisson processes; this will be revisited soon.

The rollback criterion involves a statistical test, which evaluates a used value against all the shared, estimated rates. Specifically, the confidence interval for the mean rate estimate serves as an acceptable range: $\bar{r} \pm t_{n-1,1-\alpha/2} \times S_r \big/ \sqrt{n}$ where $\bar{r}$, S_r^2, and n are the sample mean, sample variance, and sample size, respectively. The significance level α is set to 0.005. The critical value $t_{n-1,1-\alpha/2}$ is based on the Student's t distribution with $n-1$ degrees of freedom. If the used value falls outside the range, it is rejected and a rollback is triggered. Compared to the original design, this method introduces an additional parameter (i.e., the degrees of freedom) in order to adapt for various variations due to different sample sizes.

In response to rollbacks, LPs perform system state restoration. Since modeling input links involves nonstationary arrival processes, additional state information (other than flow rates) is needed. In typical discrete-event queueing simulations, processing a current arrival event includes scheduling a new arrival event, the timestamp of which relies on some "future information," i.e., in this design, future flow rates. If any pertaining future rate is later proved incorrect, a rollback is triggered; Figure 2.5 depicts such a situation where an LP is rolled back for using an underestimated flow rate during $[t, t+\Delta)$. Since resetting system state removes all the arrivals beyond t, an initial arrival has to be generated on every input link. The generating process has to consider the elapsed time from t_2 (when the last arrival occurred) to t as part of the interarrival time. Note that the new arrival must come after t because a rollback targeting at $[t, t+\Delta)$ cannot affect the system state before t.



(a) Before System State Restoration



(b) After System State Restoration

Figure 2.5: Arrival Scheduling during Rollbacks

The above issue is simplified in ad hoc queueing network simulations because the arrivals on input links are modeled as nonstationary Poisson processes. Two well-known methods for generating nonstationary Poisson arrivals are the thinning method (Lewis and Shedler 1979) and the inversion method (Çınlar 1975). The thinning method, an acceptance-rejection algorithm, requires the upper bound on the flow rate function, which is generally unavailable. Also, this method may be inefficient when the acceptance rate is low. On the other hand, the inversion method generates the arrivals times $\{t_i\}$ using a sequence of Poisson arrival times at rate 1 $\{t_i'\}$ and the expectation function of the rate function $\Lambda(t) = \int_0^t \lambda(y)dy$, as follows:

1) $u \sim U(0, 1)$, 2) $t_i' = t_{i-1}' - \ln(u)$, and 3) $t_i = \Lambda^{-1}(t_i')$.

We adopt this method because it is practical and easy to implement with one additional variable for t_{i-1}' and an array data structure for Λ^{-1} (as the rate functions in ad hoc queueing network simulations are step functions).

Naming—"Iterative" Ad Hoc Queueing Simulation Method

The term "iterative" comes from the iterative methods in computational mathematics. These iterative methods solve the problems that are formulated into the fixed-value problem $f(x) = x$, where f is a function. To find a solution of such a problem, the typical procedure of an iterative method starts with an arbitrary x_0, which is used in f to obtain $f(x_0)$; then, the value $f(x_0)$ is set to x_1. This procedure of $x_{n+1} = f(x_n)$ is repeated until the sequence $\{x_i\}$ converges. The definition of convergence varies; it can be that the difference between the last two numbers in the sequence are either zero or within a designated scale of error.

A similar phenomenon of iteration can be observed in this iterative ad hoc approach. Figure 2.6 depicts an example queueing network partitioned into two parts, each containing one node. The LPs modeling node 0 generate the state information for link B and request that of link A; by contrast, those modeling node 1 use the information for link B to produce that for link A. The relationship between the desired and the shared information can be captured by functions: F_0 for the LPs simulating node 0 and F_1 for node 1. Consider $A_{[t,\,t+\Delta)}$ as the state of link A with respect to the time period $[t, t + \Delta)$, and similarly $B_{[t,\,t+\Delta)}$ for link B; the relations can be written down as $B_{[t,\,t+\Delta)} = F_0(A_{[t,\,t+\Delta)})$ and $A_{[t,\,t+\Delta)} = F_1(B_{[t,\,t+\Delta)})$. Combining these two yields $A_{[t,\,t+\Delta)} = F_1(F_0(A_{[t,\,t+\Delta)}))$, which has the form $f(x) = x$ where f is $F_1 \circ F_0$.

Avoidance of Potential Livelocks

To prevent livelocks in this iterative ad hoc method, it is essential to comprehensively understand a physical system before building local simulation models and designing the ad hoc components, especially the rollback criteria. A livelock situation arises when two or more LPs are involved in a loop in which their shared values keep invalidating each other's simulation inputs. That is, these LPs roll back each other successively so that, from a global view, the entire simulation execution does not show forward progress. Although the zero-lookahead feature allows LPs to "bring back" others to the same simulation time, this feature cannot be blamed for livelocks. Instead, the causes are incorrect simulation models and unrealistic rollback criteria. The former one is analogous to the classical livelock problem where the application/model itself is not appropriately defined; we will not delve into this well-known issue. Instead, here we focus on the latter, which is specific to the ad hoc approach.

A feasible, legitimate rollback criterion must take into consideration the characteristics of state variables, such as randomness. For example, for a state variable with possible values ranging across some continuous space (i.e., a continuous stochastic variable), its affiliated rollback criterion needs to be flexible with regard to evaluating the "correctness" of a value. A value should be considered correct if it is within a certain range. This idea applies to discrete stochastic variables as well. However, the consequence would be more severe for continuous random variables because the probability of a continuous random variable equal to any arbitrary value is zero. This implies that, given an LP that has used a value v for some input link, another LP modeling the link is highly improbable to generate v as a state measure for the link. Hence, the LP using v is rolled back. The following example illustrates such a situation based on the queueing network in Figure 2.6.

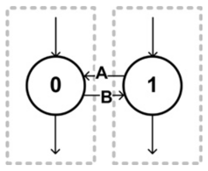


Figure 2.6: 2-Node Open Queueing Network

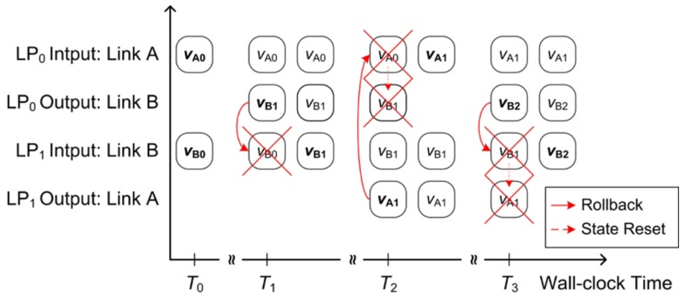


Figure 2.7: An Example of a Livelock Situation in Modeling the 2-Node Open Queueing Network in Figure 2.6

The example concerns a simulation of the queueing network in Figure 2.6 with LP_0 modeling the left part (i.e., node 0 and link B) and LP_1 covering the right one. LP_0 requires the state information of link A as input to its local simulation model, and this information is shared by LP_1. Similarly, LP_1 relies on LP_0 for link B. The link states are measured by the flow rates estimated over Δ seconds, and the LPs must use the exact value their corresponding LP generates. Considering a time interval $[t, t + \Delta)$, the process used by the two LPs to reach an agreement on the rates is depicted in Figure 2.7; clearly, they fail and fall into a livelock. The detailed process is as follows:

1. At wall-clock time T_0, both LP_0 and LP_1 have not generated the flow rates for their corresponding links with respect to $[t, t + \Delta)$. As a consequence, they adopt arbitrary values: LP_0 applies v_{A0} for link A and LP_1 uses v_{B0} for link B.
2. At time T_1, LP_0 produces the estimated flow rate of link B, v_{B1}, which rolls back LP_1.

3. Then at T_2, LP_1 observes the flow rate of link A being v_{A1} after using v_{B1} for link B. As a consequence, LP_0 is rolled back and its shared state information about link B, v_{B1}, is revoked.
4. At time T_3, a similar situation occurs: LP_0 derives a new value for link B, v_{B2}, due to the usage of v_{A1}. Rolling back LP_1 results in the simulation ending up in a situation resembling that at T_1.

This loop of rollbacks is anticipated to continue because the flow-rate estimates are continuous stochastic variables; it is highly unlikely that $v_{Ai} = v_{Ai+1}$ (nor $v_{Bi} = v_{Bi+1}$) for any integer $i \geq 0$.

2.4 Experiments and Results

This section evaluates the iterative ad hoc queueing simulation method with three experiments. The first two methods revisit those discussed earlier but use the new iterative ad hoc method. The third one loads the previous tandem network with heavy traffic. All three are to show that the method is effective in terms of identifying system transients. The metrics for evaluation include the flow rates of input links and the mean queue lengths at nodes. The results are compared with the sequential counterparts.

Welch's *t* Test

The following experiments use Welch's *t* test to statistically assess the (null) hypothesis that the mean values of two samples are equal. This test modifies the well-known Student's *t*-test to consider that two samples may have unequal variances. Its statistic *t* and the degrees of freedom *v* are defined by Equation (1) where $\overline{X}_i$, S_i^2, and N_i are the *i*-th sample mean, sample variance, and sample size, respectively. Given a significance level α (e.g., 0.01), the hypothesis is rejected if the derived *p*-value is below α; the *p*-value for a two-tailed test based on Student's *t*-distribution is $1 - (F_v(|t|) - F_v(-|t|))$ where F_v denotes the respective distribution function.

$$t = \left(\overline{X}_1 - \overline{X}_2\right) \Big/ \sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}} \quad \text{and} \quad v = \left(\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}\right)^2 \Big/ \left(\frac{S_1^4}{N_1^2(N_1-1)} + \frac{S_2^4}{N_2^2(N_2-1)}\right) \tag{1}$$

The two samples in the subsequent hypothesis tests are 1) the data from the ad hoc runs and 2) those from the sequential counterparts. Regardless of the metric, the size of the ad hoc sample (i.e., N_1) is always 10, same as the number of replicate ad hoc runs, and that of the sequential sample (i.e., N_2) is 100.

Experiment 1: Partially-Bidirectional Tandem Network

This experiment concerns the network in Figure 2.2(a) with all the configurations from Section 0. The simulation results based on the iterative ad hoc method are shown in Figure 2.8. Regardless of the Δ value, the ad hoc simulations perform well on capturing the estimated flow rates and mean queue lengths. The revealed choppiness in estimates is anticipated due to the randomness in the simulation models. Furthermore, it is expected that a large Δ may weaken the real-time response as the changes are averaged out across the entire update periods. However, such issue is insignificant in this experiment.

Figure 2.9 plots the *p*-values from Welch's *t* tests on the same two measurements of interest. These results reaffirm the good performance of the iterative ad hoc method because the vast majority of the *p*-values are above the critical level α = 0.01. Some exceptions include those at simulation times 3:54, 4:16, 4:28, and 4:32; they are rather insignificant because 1) the one at time 3:54 occurs just prior to the arrival rate increase at time 4:00 and 2) they are sparse across the entire simulation.

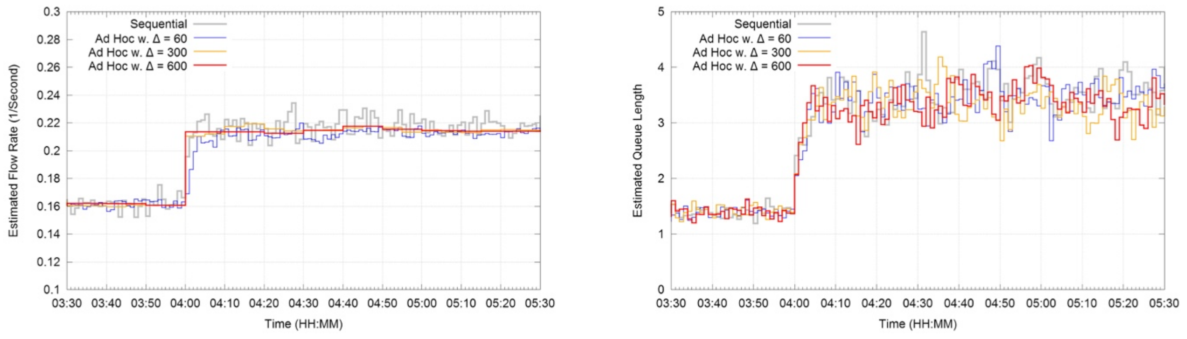
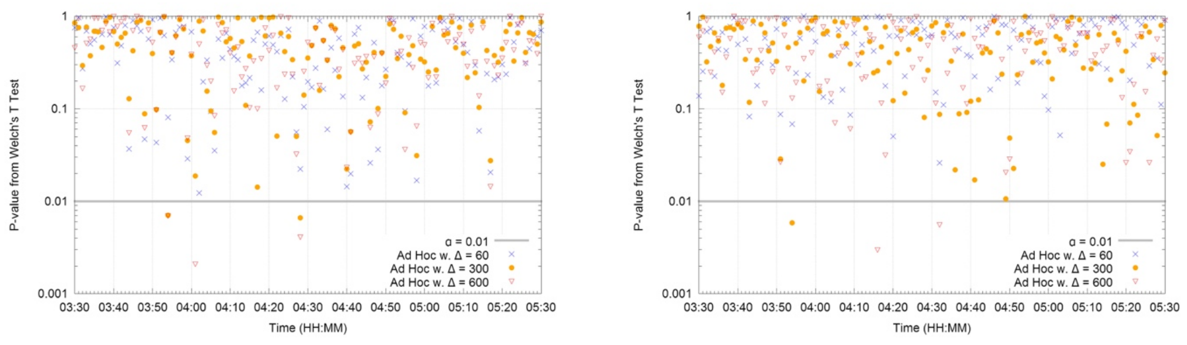


Figure 2.8: Estimated Arrival Rate across Link 10 and Mean Queue Length at Node 4 in Experiment 1



(a) Estimated Arrival Rate across Link 10 (b) Estimated Mean Queue Length at Node 4

Figure 2.9: *P*-Values from Welch's *t* Tests in Experiment 1

Experiment 2: Completely-Bidirectional Tandem Network with Moderate Traffic

This experiment extends that in Section 2.2 by replacing the original ad hoc method with the new one. The network of interest is in Figure 2.2(b) and the results are in Figure 2.10. Here (and afterwards), the flow-rate metric is skipped as it leads to the same conclusion as the queue-length metric—that is, the iterative ad hoc method performs well. Nevertheless, again, sporadic hypothesis rejections occur but they are not major, as those in the previous experiment.

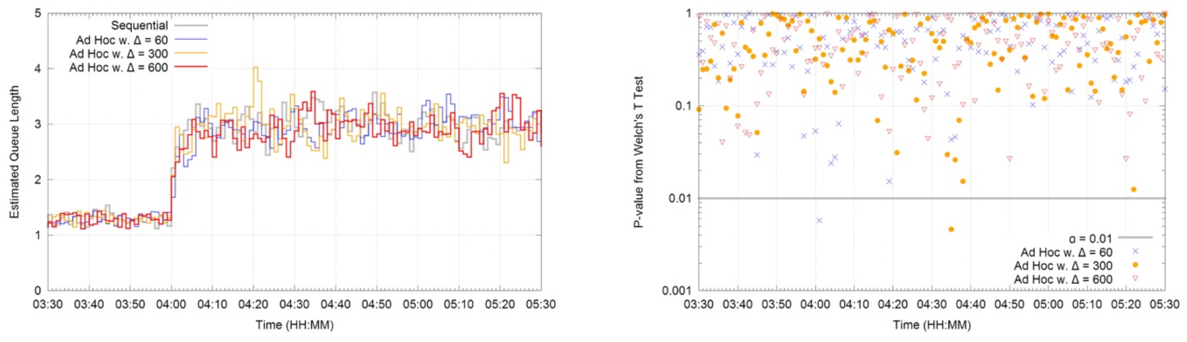


Figure 2.10: Point Estimates and *p*-Values on Estimated Mean Queue Length at Node 4 in Experiment 2

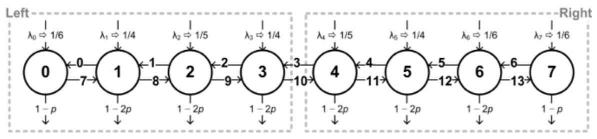

Left
Right
$\lambda_0 \Rightarrow 1/6$
$\lambda_1 \Rightarrow 1/4$
$\lambda_2 \Rightarrow 1/5$
$\lambda_3 \Rightarrow 1/4$
$\lambda_4 \Rightarrow 1/5$
$\lambda_5 \Rightarrow 1/4$
$\lambda_6 \Rightarrow 1/6$
$\lambda_7 \Rightarrow 1/6$
$1-p$
$1-2p$

Figure 2.11: An 8-Node Completely-Bidirectional Tandem Network with Heavy Traffic

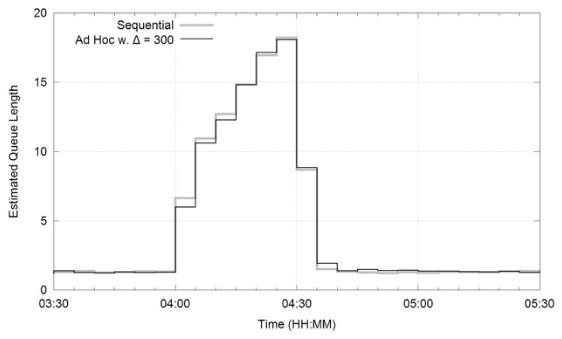


Figure 2.12: Estimated Mean Queue Length at Node 4 in Experiment 3

Experiment 3: Completely-Bidirectional Tandem Network with Heavy Traffic

This experiment intends to show that the iterative ad hoc method is capable of promptly responding to sudden, massive state changes. The network of interest is depicted in Figure 2.2(b) and the system transients are introduced by increasing the external arrivals to each node as depicted in Figure 2.11. This makes node 3 saturated—an extremely busy server with its queue building up. Node 4 is also saturated with its steady-state traffic intensity reaching approximately 0.97. Since the queues grow rapidly, the system transient period is configured to last for only 30 minutes. This allows the ad hoc method to demonstrate that it can also capture state changes in the opposite direction (i.e., decrease in flow rates).

The simulation results affirm that the iterative ad hoc method is competent in this heavy-traffic scenario; see Figure 2.12 for the estimated mean queue length at node 4. For clarity, Δ is set to a moderate value (Δ = 300), leaving off the jagged results from small Δ. Further, the respective *t* tests indicate insignificant differences between the ad hoc and sequential simulations (the figure is not shown due to space constraints).

2.5 Conclusions

By extending the work by Huang et al. (2012), which studies the prediction accuracy of steady-state metrics, this chapter demonstrated the competitiveness of ad hoc distributed simulation on modeling short-term system dynamics in opening queueing networks. First, we conducted two experiments with increases in flow rates during simulation executions to argue that the original ad hoc method in the previous work fails to immediately reflect system transients. The extent of the delayed response relates to the length of the update period Δ. The failures are due to the fact that the state updates shared by individual LPs are regarded as predictions of future system states, rather than reflections of the current state.

To address the delayed-response issue, we proposed an iterative ad hoc method along with detailed discussions on the design of the relevant components as well as the potential livelock issue arising in certain simulation models. The new method was empirically evaluated by three experiments that demonstrated the potential of the proposed iterative ad hoc approach to capture system dynamics. The evaluation was based on point estimation, which includes not only the direct comparison of the averaged statistics but also Welch's t test to provide more compelling and comprehensive statistical evidence.

3 Online Calibration of Traffic Simulation Models

Efforts to address operational issues in transportation are ongoing and are the focus of many research efforts. A number of these efforts are geared toward developing microscopic traffic simulation models to accurately represent the complex and dynamic operation of a transportation network. One of the challenges with such models is that they do not always adequately reflect field conditions – particularly when representing traffic operations across different time periods. This chapter presents a robust calibration procedure that aims to increase the accuracy of calibrated microscopic traffic simulation models while underscoring the case of online calibration. This procedure is based on a Monte Carlo approach to generate candidate parameter sets, which are aimed to produce calibrated simulation models. Model runs of these parameter sets are evaluated against a robust calibration criteria including startup and saturation flow characteristics and travel time distributions. The parameter sets that satisfy these criteria are considered suitable to produce adequately calibrated models that are capable of accurately reflecting field performance measures. In applying this procedure, the results indicate that it is capable of producing well calibrated models, while highlighting the need for online calibration procedures to be applied to models that are meant to represent traffic in real-time, or across multiple time periods. The results also suggest that this approach offers a robust and effective method of calibrating simulation models where disaggregate level vehicle data are available – which is becoming more prevalent with further advancements in mobile sensor and connected vehicle technologies.

3.1 Introduction

Traffic congestion incurs a cost of an estimated one hundred billion dollars each year in the United Sates (Schrank et al 2011). Addressing this high cost of congestion is one of the primary issues facing many of today's urban and suburban areas. However, as resources have become increasingly constrained, and right-of-way and construction costs have increased, there has been a fundamental shift in the manner in which congestion issues are addressed. This shift is reflected in the substantial efforts to develop and implement alternate means for alleviating congestion. These solutions utilize advanced technologies to increase the efficiency of today's transportation network.

Microscopic traffic simulation is being viewed as a tool that is able to increase the capabilities of these advanced transportation solutions. However, simulation models must be appropriately calibrated to provide results that accurately reflect field performance measures (Henclewood, 2012). To date, the predominant means of calibrating a microscopic simulation model is based on selecting a set of calibration parameters that allows the model to reflect the average field performance measures, not the field performance measure distribution (Henclewood, 2012). Additionally, such models are calibrated with respect to particular time periods and therefore limit their applicability and level of accuracy when modeling time periods outside those for which they were calibrated. To highlight the importance of proper calibrations, this chapter presents a statistical calibration method that offers a more effective method of calibrating these models, where significant quantities of vehicle trajectory or individual vehicle travel time data (e.g. from GPS probe vehicles or video-based techniques) are available. This calibration method coupled with individual vehicle data has the potential to improve the assignment of model parameter values and increase the likelihood of producing robust and precise traffic simulation models. This chapter also presents a case for real-time/online calibration in light of the results from this robust calibration procedure.

The following section presents a brief literature review of previous calibration efforts related to microscopic modeling of surface transportation. Next, the proposed statistical calibration method and the results of its application to the Next Generation Simulation (NGSIM) dataset are presented. Lastly, a discussion regarding the need for temporal application of the developed calibration procedure is presented.

3.2 Related Work

A variety of methods have been developed to calibrate traffic simulation models. Hollander and Liu presented a comprehensive review of current calibration methods and highlighted the fundamental requirements for calibrating simulation models (Hollander and Liu 2008). Zhang and Ma (2004) also reviewed the current calibration methods and grouped them into three categories, 1) trial-and-error heuristics, 2) genetic algorithms, and 3) simulated annealing (Zhang and Ma 2004). They found that a majority of the calibration methods were either trial-and-error heuristics or genetic algorithms methods, with the majority being trial-and-error heuristic methods.

A few notable trial-and-error heuristic methods are highlight below. These include Chu et al.'s (2004) four-step calibration process. These four steps include determine a suitable driver behavior model(s), select an appropriate route choice mechanism, estimate origin-destination pairs accurately and then fine tune the model to reflect field performance measures. Oketch and Carrick (2005) also developed a trail-and-error heuristic method that is based on 1) determining proper model parameter values such as, aggressiveness, awareness, target headways and reaction times and 2) estimating representative origin-destination matrices. The trial-and-error method that Toledo et al. proposed is an iterative calibration method to jointly estimate origin-destination flows and values for behavioral parameters (Toledo et al., 2004). Dowling et al.'s (2002) method included a series of four step processes to calibrate a model, which included error checking, calibration for capacity, calibration for demand, and overall review (Dowling et al. 2004).

As for calibration methods based on genetic algorithms, two of the more noteworthy applications were presented by Park and Won (2006), and Zhang et al. (2008). In addition to numerous calibration methods, various criteria have been developed to determine when a model is properly calibrated. Hellinga (1998) found that the criteria tend to be subjective due to their dependence on what is being modeled and the goals of the study. Hollander and Liu presented a summary of a number of these calibration criteria in (Hollander and Liu 2008). To date, one of the main criteria to determine whether or not a model is calibrated involves a parametric, first moment statistical comparisons of field and simulated performance measures (Henclewook 2012). For instance, Park and Schneeberger (2003) used results from a t-test to compare simulated and field travel time averages as the criterion to determine when a model is calibrated. Park and Won (2006) developed a criterion that designated a model to be calibrated when the model's travel time distribution "includes the entire field-measured values".

Although these criteria may be sufficient to evaluate general traffic performance at an aggregated level, it is questionable if such means can represent traffic performance at an individual vehicle level. This chapter highlights the importance of proper calibrations by providing a statistical calibration procedure that allows the corresponding simulation to accurately represent the underlying distributions that characterize traffic at both an aggregated and an individual vehicle level.

3.3 Statistical Calibration Procedure

The proposed calibration method includes a comprehensive evaluation of the selection of calibration parameter values and a two-part criteria process (Figure 3.1). This calibration is not intended to address issues related to model construction but instead underlying model parameter value selection.

First, a Monte Carlo approach is employed to generate potential parameter sets for calibrating a traffic simulation model. After identifying effective parameter sets for calibration, simulations with these parameter sets are evaluated against a robust set of calibration criteria to determine which are calibrated. There are two steps to the calibration parameter value selection process: 1) evaluation of startup and saturation flow and 2) statistical evaluation of the performance measure distributions (travel time in this example) which includes the application of the Wilcoxon-Mann-Whitney test, the Kolmogorov-Smirnov test, and a heuristic form fit test. The parameter sets that satisfy both these steps are considered as adequately calibrated.

The following will detail the various components of the calibration procedure.

Monte Carlo Process – Identifying Effective Calibration Parameters

This Monte Carlo process is expanded from (Miller 2009 and Roess et al. 2003). This process consists of three steps: 1) performance measure and initial parameter selection, 2) Monte Carlo simulations, and 3) parameter elimination. Steps 2 and 3 are repeated until a pre-defined stopping condition is satisfied.

In the first step, the performance measure for the model is selected. The performance measure can be any number of, and any combination of, travel times, flows, delay, and queue lengths. For the initial parameter selection, certain parameters can be eliminated immediately, for example, parameters that are not used in the model due to the facility type being modeled (i.e. arterial versus freeway). Miller (2009) demonstrated that approximately half of VISSIM's 50 calibration parameters could be eliminated in this step. For each of the selected parameters, a reasonableness range is determined.

Step two involves a Monte Carlo experiment that generates a series of simulation runs used to isolate the sensitivity of the performance measure to the parameter values. One thousand parameters sets are created in a Monte Carlo fashion, by repeated random selection of parameter values using uniform distribution. The value for each of these parameters is limited by its reasonableness range, determined as part of step 1. These parameter sets are then used as inputs to generate 1000 unique simulation runs where only these parameter values vary between runs. From each simulation run, the selected performance measures are extracted and analyzed. Note, replicate runs were not conducted for each model as employing the Monte Carlo method to create these models appropriately approximates the effect of executing replicate runs of each model – on an aggregate level. Step three compares the performance measures with the values for each parameter and eliminates parameters whose values have little or no effect on the measures. Default values are assigned to the eliminated parameters and step two is repeated until no parameters can be eliminated due to a lack of influence on the performance measures. Figure 3.2 illustrates the execution of the process.

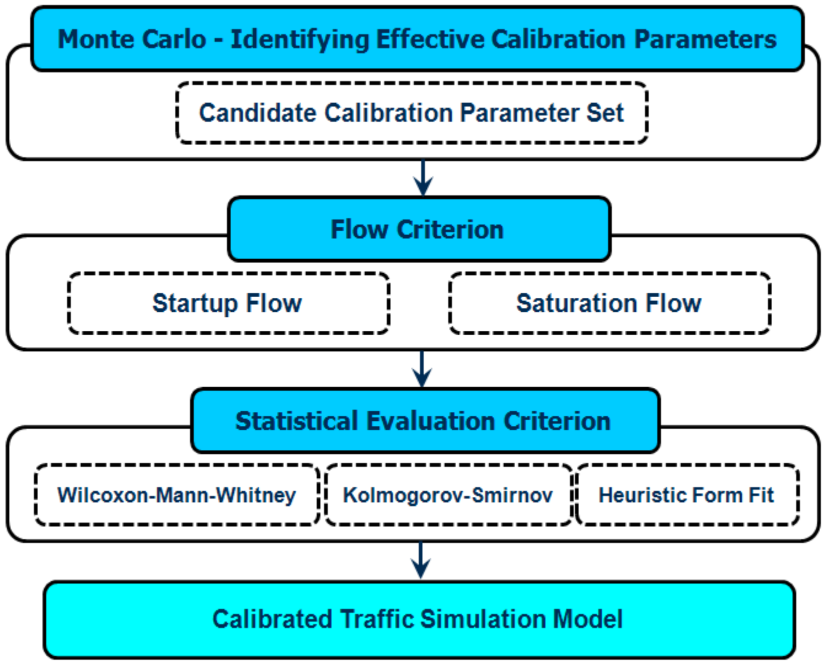

Monte Carlo - Identifying Effective Calibration Parameters
Candidate Calibration Parameter Set
Flow Criterion
Startup Flow
Saturation Flow
Statistical Evaluation Criterion
Wilcoxon-Mann-Whitney
Kolmogorov-Smirnov
Heuristic Form Fit
Calibrated Traffic Simulation Model

Figure 3.1: Statistical calibration procedure.

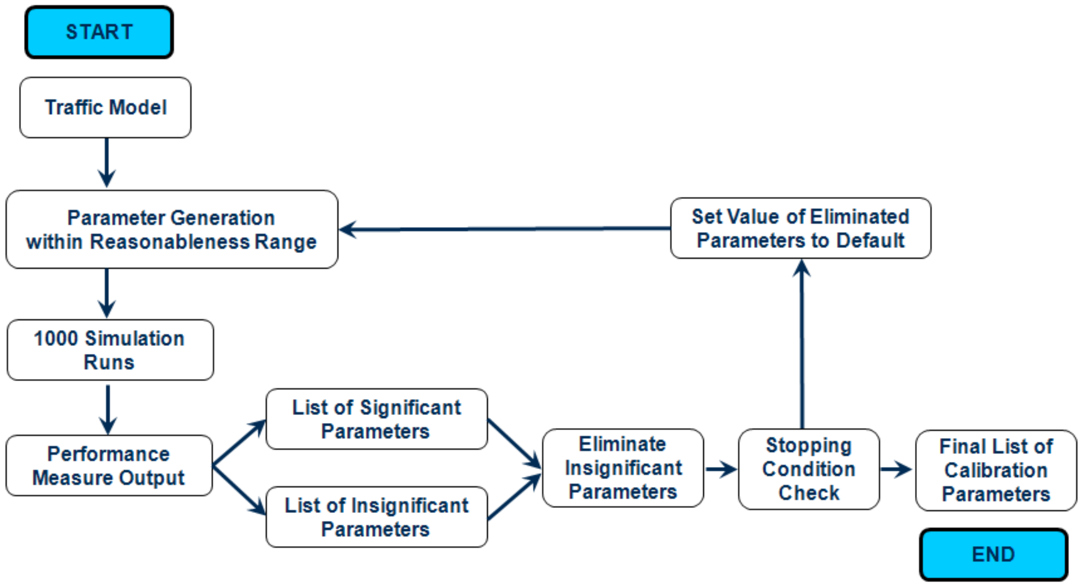

START
Traffic Model
Parameter Generation within Reasonableness Range
Set Value of Eliminated Parameters to Default
1000 Simulation Runs
Performance Measure Output
List of Significant Parameters
List of Insignificant Parameters
Eliminate Insignificant Parameters
Stopping Condition Check
Final List of Calibration Parameters
END

Figure 3.2: Monte Carlo calibration parameter selections.

Startup and Saturation Flow Criteria

This criterion is the first part of two step criteria process to select calibrated parameter sets. This part compares field and simulated saturation flow rates to ensure that the models produce reasonable estimates. This is an imperative step as it is possible for models to produce accurate estimates of performance measures, such as travel time, while over or under estimating saturation flow rates. Failure to adequately model saturation flow could prove significant. For example, by overestimating saturation flow a simulation will overestimate capacity. In a scenario analysis where base volumes are increased to a higher level to represent build conditions it is possible they could exceed field capacity (implying significant congestion) but the simulation would continue to show uncongested operations. As a result, the simulation would no longer reflect field conditions.

Saturation headway is defined as "the amount of time that a vehicle in the stopped queue takes to pass through a signalized intersection on the green signal, assuming that there is a continuous queue of vehicles moving through the intersection" or "the constant headway achieved is referred to as the saturation headway, as it is the average headway that can be achieved by a saturated, stable moving queue of vehicles passing through the signal" (Henclewood 2012). Headway measurements of the first four vehicles are not considered when estimating saturation headway. However, these headway measurements were included (referred to as startup flow) in this effort to allow for a calibration of parameters related to queue start-up as well as saturation flow, providing a better estimate of capacity (Henclewood 2012). These startup and saturation flow criteria not only facilitate greater confidence in the results from a calibrated model but also provide some level of protection from the potential dangers in the application and implementation of the calibrated model. Startup and saturation flow are measured from headway measurements at an intersection. Startup flow is measured from the second vehicle in the queue to fifth vehicle, while saturation flow is estimated based on measurements from the sixth to the ninth vehicle. Headway measurements are then averaged for each cycle and converted to startup (ssu) and saturation (s) flows via (1) (Miller et al. 2012). The startup and saturation flow rates (per cycle) are then used to create frequency density plots of the observed flow rates.

$$s_{su} = \frac{3600}{h_{su}} \qquad s = \frac{3600}{h} \tag{1}$$

where: s_{su} = startup flow (vehicles/hour)
h_{su} = startup headway (second)
s = saturation flow (vehicles/hour)
h = saturation headway (second)

To create the startup and saturation flow criterion, a reasonable range is chosen to aid in the evaluation of whether or not a parameter set may be retained for further consideration in a calibrated model. The reasonable range is constructed by forming a 95% confidence interval around the mean flow values from the field. If field data are insufficient to construct an appropriate confidence interval, it is recommended that a bootstrap approach be used to bolster the field data that will be used to make inferences about field startup and saturation flow estimates (Henclewood 2012). Bootstrapping is a means of making statistical inferences in the presence of limited data. The bootstrap method involves the re-sampling of data, with replacement, in order to generate an empirical estimate of the entire sampling distribution of a statistic (Ross et al. 2003).

Statistical Evaluation Criterion

For the models producing startup and saturation flow measurements within a 95% confidence interval, a statistical evaluation criterion is examined. Since field data often do not fit known distributions, non-parametric tools are used to ensure proper comparisons of field and simulated data. Two sets of non-parametric tools were used to establish the statistical calibration criterion. The first set of tools employed the use of the Wilcoxon-Mann-Whitney (WMW) test to conduct a general distribution comparison to primarily determine the homogeneity between field and simulated performance measures (travel time). The second set of tools involves the use of a more stringent comparison of the distribution of performance measures. This set of tools employ the use of the Kolmogorov-Smirnov (KS) and heuristic form fit (HFF) tests. Both sets of tests complement each other. Each test attempts to satisfy the shortcomings of the other which were in part due to some of the fundamental principles underlying their original development. For a full discussion regarding the selection of each test and how they are able to complement each other, readers are encouraged to review the works presented in (Henclewood 2012) and (Kvam and Vidakovic 2007).

The WMW test was chosen to evaluate the general differences between field and simulated travel time distributions. One of the outputs is a p-value that is used in the decision to accept or reject the null hypothesis H_0 (for this case that the field and the simulated travel time distributions are equivalent). A calibration criterion based on the WMW p-value is the rejection of H_0 , and subsequently a model parameter set, when the p-value is ≤ 0.01.

This criterion is then paired with another that is based on a more stringent comparison of the travel time distributions. The need for a more stringent comparison of travel time distributions is due to the nature of the WMW test that does not explicitly consider the absolute magnitude of the differences in travel time. Given the multimodal nature of the actual travel time distributions, the KS non-parametric test was selected to further compare the distribution of travel time datasets. Unlike the WMW test, the KS test does take into account the magnitude of the differences between the data points of the samples being compared. To formalize the calibration criterion based on the KS test, a model comparison whose test statistic corresponds to a p-value of ≤ 0.01 will result in a rejection of H_0.

The null hypothesis in this case states that there is insufficient evidence to suggest that a parameter set's simulated distribution of travel time estimates is different from the same distribution obtained from the field. The rejection of H_0 for a particular travel time segment removes that model from being considered as a possible calibrated model for a particular period and travel direction.

The HFF test (developed for this effort) was also included to provide an alternative distribution comparison method. The HFF test is devised to compare the rate of change of the CDFs (instantaneous density) of the two distributions. A test statistic, H, is created and is defined as the sum of squares of the difference between the rate of change between the CDF of the field and simulated data, $F_n(x)$ and $\dot{F}_n(x)$ respectively. Mathematically;

$$\boldsymbol{H} = \sum \left(\frac{\boldsymbol{dF_n(x)}}{\boldsymbol{dx}} - \frac{\boldsymbol{d\dot{F}_n(x)}}{\boldsymbol{dx}} \right)^{\boldsymbol{2}} \tag{2}$$

While there are no special assumptions associated with the HFF test, the method does not take into account shifts along the x-axis, i.e. differences in central tendencies (mean/median). This is concerning as for a given comparison, H having a value equal to or close to zero does not necessarily mean that the results from a parameter set fits field data. The only definitive statement that may be made is that the shapes, or forms, of the two distributions, including its modal characteristics, are similar. Since there is currently no p-value for this test statistic, parameter set simulations that produce H values in the bottom half of the range of H-values are considered as potential calibration model candidates.

The statistical evaluation criteria use the WMW, KS and HFF tests to examine the field and simulated travel time distributions. All three statistical tests are used to determine which parameter sets satisfy the statistical evaluation criteria. The following steps outline the application of these tests and how parameter sets that satisfy the statistical evaluation criteria are selected:

1. Conduct the WMW test and retain parameter set simulations whose WMW test yielded p-values = 0.01. This will be a set of parameter sets denoted by Mn|U.
2. Conduct the KS test and retain parameter set simulations whose KS test yielded p-values = 0.01. This will be a set of parameter sets denoted by Mn|D.
3. Conduct the HFF test and retain parameter set simulations whose H value is in the bottom half of the range of H-values. This will be a set of parameter sets denoted by Mn|H.
4. To obtain the set of parameter set simulations that satisfy the statistical evaluation criteria, carry out the following set operation

$$\boldsymbol{M_f = M_n|U \cap (M_n|D \cup M_n|H)} \tag{3}$$

The parameter set simulations that satisfy both the KS and HFF tests are combined as they evaluate the same characteristic of the data – the shape of its distribution. The union of these two sets facilitated the inclusion of simulated data that may have been otherwise excluded. The exclusion of such datasets may have been the result of a violation of a KS test assumption and/or the inability to provide a large enough p-value to be included in Mn|D, despite having similarly shaped distributions. In other words, Mn|H and Mn|D are combined to further minimize the probability of committing a Type I error, incorrect rejection of a true null hypothesis, regarding distribution shapes. With these tools working in tandem, parameter set simulations that produce accurate estimates of field performance measure distributions will be considered as calibrated replicates.

3.4 Applying the Calibration Procedure

The proposed statistical calibration method was applied to the data collected as part of the NGSIM program (FHWA 2007). This application is intended to investigate the feasibility of the proposed statistical calibration method when individual vehicle and network operation data are available. The NGSIM data were collected on November 8, 2006, between 12:45PM and 1:00PM (referred to as Noon period in this xhapter) and 4:00PM and 4:15PM (referred to as Evening period in this chapter), along Peachtree Street in Atlanta, Georgia. This data set consists of trajectory information (with a resolution of a tenth of a second) for all vehicles traversing the corridor during the study period. In addition, signal phase information at each intersection, origin/destination (OD) data for each vehicle, turning movement distribution data at each intersection, and a series of other traffic related information were also collected.

For the method application, a detailed VISSIM model of the study area was created. VISSIM is a discrete, stochastic, time step based microscopic simulation model. In this model all vehicles are modeled individually, based on a psycho-physical driver behavior model developed by Wiedemann (PTV 2011). Several verification iterations were completed to ensure that the model correctly represented the study area, as well as the traffic operations during the study period. A VISSIM trip-chain file was created to drive the simulation model by dictating when vehicle entered the simulated network and their origin-destination (O-D) information.

Uncalibrated Model

After inputting the necessary field data from the NGSIM study area into the VISSIM model, ten replicate runs were executed. The results from these runs were then used for the comparison between field and simulated performance measures. In examining the average of the 10 replicate runs there were some discrepancies between the simulated and field travel time estimates. A common discrepancy is that VISSIM tends to underestimate field travel times. The smallest and largest differences between VISSIM and field travel time are approximately 8 seconds (Noon-Southbound) and 33 seconds (Evening-Southbound). When comparing standard deviations, it is seen that the values produced by VISSIM are similar to those from the field, which indicates that VISSIM's approximation of the travel time variation estimates is rather similar to that of the field. Travel time frequency density plots are provided in Figure 3.3. The plots of the simulated travel times generally capture the bi-modal or tri-modal form of the field travel times. The differences between the plots tend to be a shifting of the centroid of the modes or proportionality between the different modes. However, in all cases the general form of the distribution is reflected, likely indicating many of the differences may be addressed through a calibration procedure.

Procedure Application - Monte Carlo Process

Based on the procedure described earlier, a Monte Carlo approach was applied to create candidate parameter sets that sufficiently represented the sample space for each parameter. Ten final parameters were selected to calibrate the NGSIM model. The selected parameters are average standstill distance (average desired distance between stopped cars), additive part of safety distance, multiplicative part of safety distance (two parameters which have major influence on the safety distance and thus affect the saturation flow rate), maximum deceleration (own), maximum deceleration (trailing) (two parameters which model the aggressiveness of lane changing behavior), minimum headway (front/rear) (minimum distance to the vehicle in front that must be available for a lane change in standstill condition), safety distance reduction factor (a parameter to takes effect for the safety distance of the trailing vehicle for the decision whether to change lanes or not, and the own safety distance during a lane change), maximum deceleration for cooperative braking (a parameter which defines if a trailing vehicle will start cooperative braking allowing a leading vehicle to change from an adjacent lane into its lane), lane change distance and desired speed distribution range (Miller 2009). The Monte Carlo process produced 1000 unique parameter sets and each set was simulated in the VISSIM model. Travel time (Figure 3.4) and startup and saturation flow measures were extracted from the simulation runs.

Procedure Application - Startup and Saturation Flow Criteria

Travel time and saturation flow measures were extracted from the simulation runs with the 1000 unique parameter sets in the Monte Carlo process. These measures were then analyzed to determine which combination of parameter values most closely reproduced the NGSIM results. From the NGSIM video, a 95% confidence interval was constructed using the bootstrapped flow measurements. This confidence interval was produced by the percentile-t method and was selected as the reasonable range for simulated flow rates. The green horizontal lines in Figure 3.5 represent the percentile-t 95% confidence interval for the startup flow (left) and saturation flow (right), while the red line represents the respective mean value. The upper and lower bounds for startup flow is 1342 and 1488 veh/hr/ln respectively; while for saturation flow the bounds are 1499 and 1796 veh/hr/ln. In applying the flow criterion, simulated models that produced average startup and saturation flow measurements that are within the respective confidence intervals are retained for further consideration as calibrate models. After applying this criterion, of the 1000 VISSIM models only 159 produce flow measurements that are within the 95% confidence intervals. These 159 models are examined against the statistical evaluation criteria to determine which models most closely simulate the field results.

Procedure Application - Statistical Evaluation Criteria

For statistical evaluation criteria the WMW, the KS and a HFF tests were applied on the 159 parameter set simulations that satisfied the above saturation flow criteria. These tests were performed for each time period and direction of travel in the NGSIM data set (Noon and Evening, Northbound and Southbound). Ideally, the parameter sets for each of these time periods and directions should be the same. However, the above analysis yielded a number of different calibrated parameter sets across periods and directions. This means, for each time period and direction, different parameter sets were able to produce a calibrated model.

Figure 3.6a presents travel time frequency density plots from this single calibrated parameter set (red line), as well as plots from the field data (black line) and the original VISSIM model with default parameter values (yellow line). For comparison, Figure 3.6b and 3.6c show the startup and saturation flow plots for the same parameter set respectively. The selected parameter set is deemed properly calibrated.

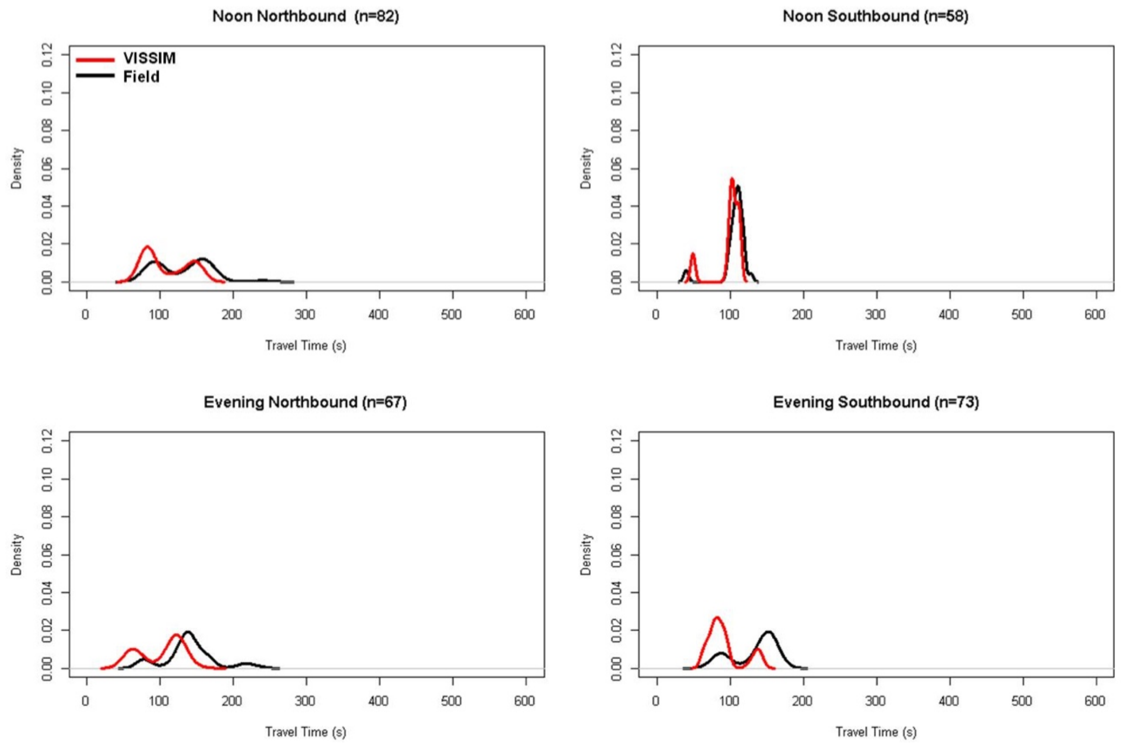


Figure 3.3. Travel time frequency density plots of NGSIM Field vs. VISSIM (single run).

3.5 Discussion

The two-part criteria process was carried out for each time period and direction of travel in the NGSIM data set (12PM and 4PM, Northbound and Southbound). Using only the parameter set simulations that satisfy part one of the calibration criteria, the next set of parameter sets simulations were eliminated after applying the WMW test. The remaining parameter set simulations, Mn|U, had sufficiently similar distributions that there was insufficient evidence to reject H0. Similarly, the KS and HFF tests were conducted. The final set of parameter set simulations that were obtained after applying the above set operation. The remaining set of parameter simulations are deemed calibrated – according to the two-part criteria process. Table 3.1 presents a summary of the number of parameter set simulations that were retained as the various calibration criterion were applied.

A final set of calibrated parameter sets must next be selected based on comparisons with four different field datasets – two periods with two travel directions. The preceding analysis resulted in a selection of parameter sets that are adequately calibrated for each time period – direction alternatives, Mf|12/NB, Mf|12/SB, Mf|4/NB, and Mf|4/SB, where the subscript represents time period and direction (i.e. Mf|12/NB represents the set of adequately calibrated replicates for the 12PM, northbound traffic). Ideally, the parameter sets for each of these time periods and directions should be the same. However, the above analysis yielded a number of different calibrated parameter sets across periods and directions. This means, for each time period and direction, different parameter sets were able to produce a calibrated model. Table 3.2 presents the number of parameter sets that are the same for different time periods and travel direction. From the table below, one observes that there were 43 simulation models that were calibrated for Noon period, and two for the Evening period. And there was only one model that was calibrated for both periods and travel directions.

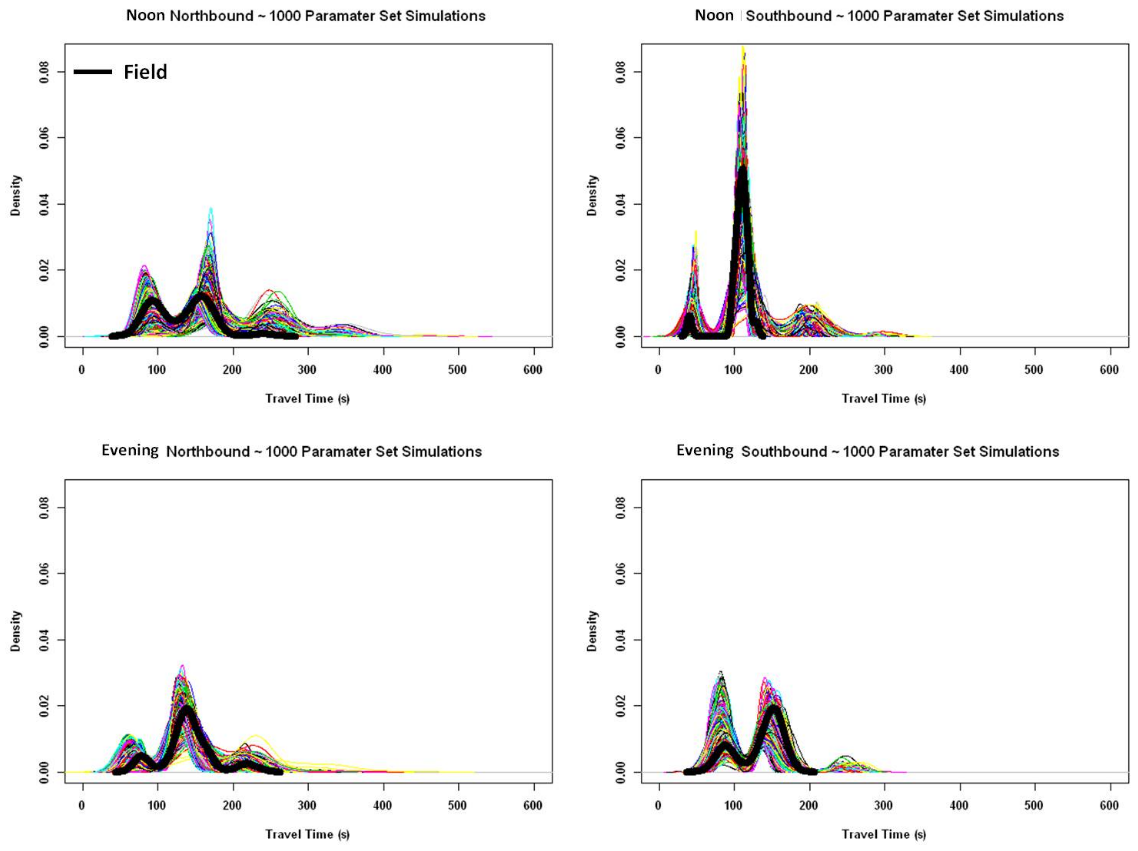


Figure 3.4. Travel time frequency density plots of NGSIM field vs. 1000 VISSIM runs.

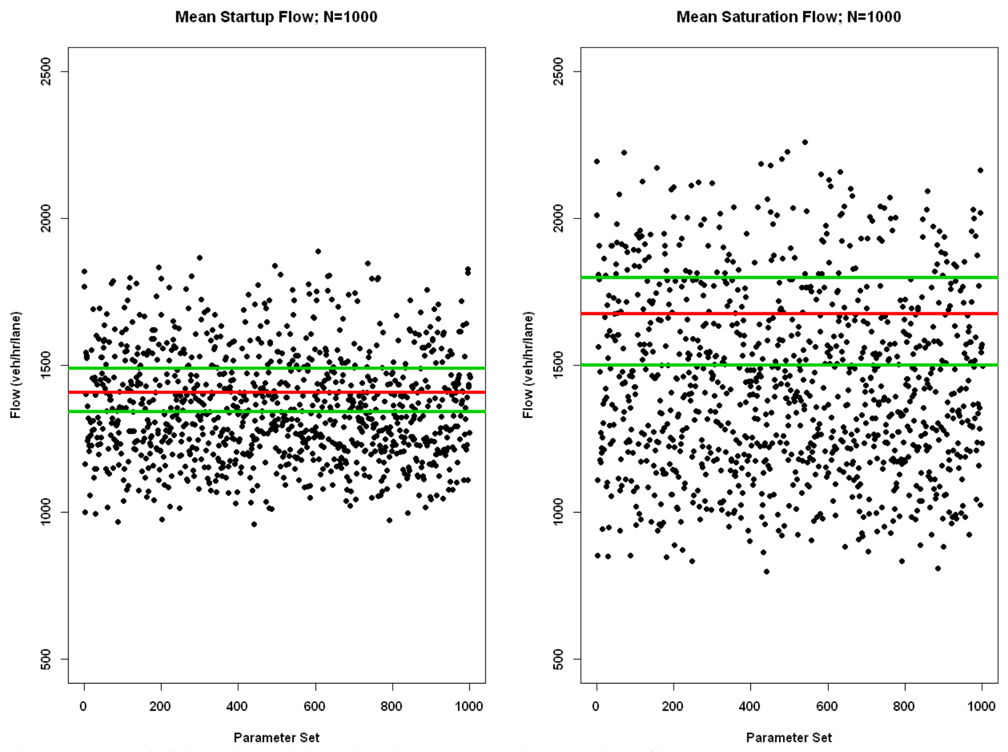
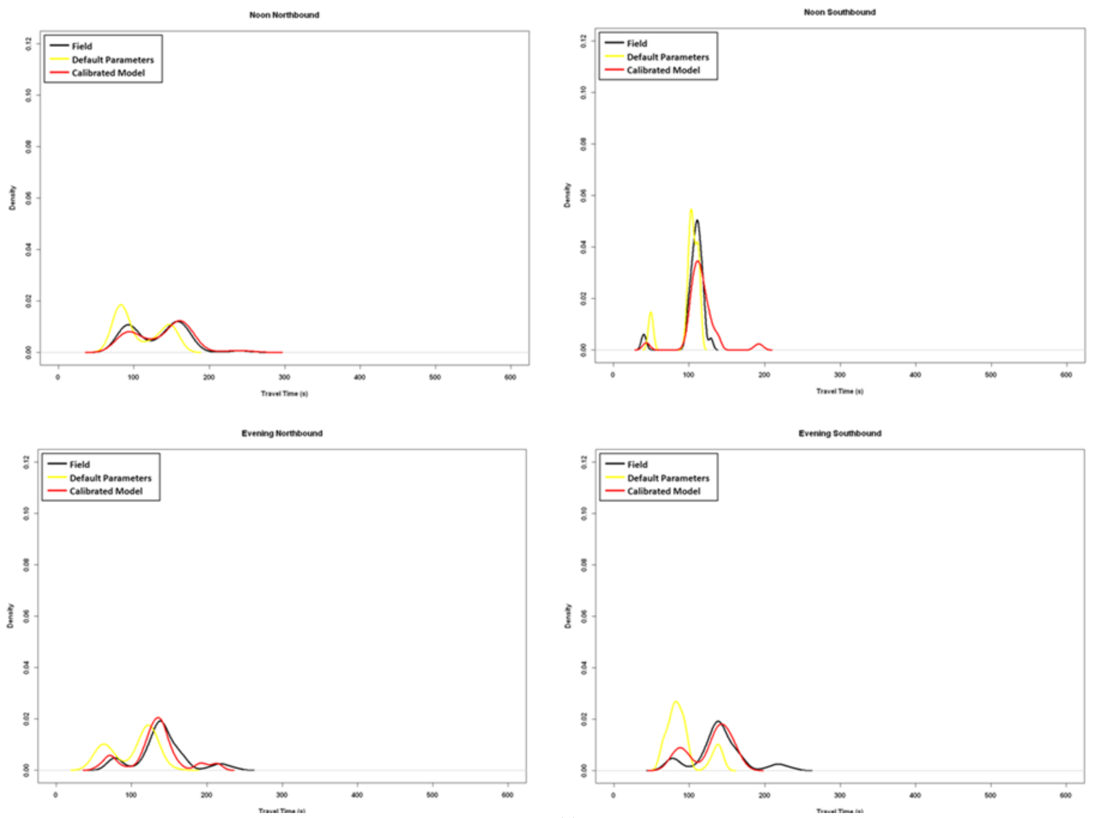
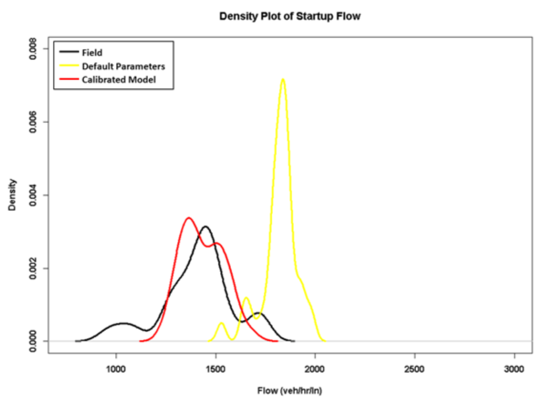
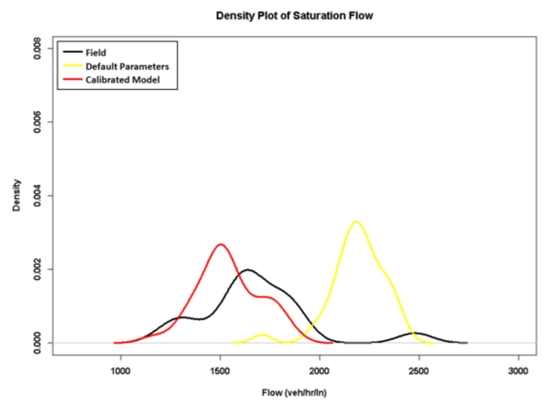


Figure 3.5: Field (red) and simulated startup and saturation flows.



(a)





(b) (c)

Figure 3.6: NGSIM field, VISSIM default parameter, and calibrated model comparison: (a) travel times, (b) startup flow rate, (c) saturation flow rate.

TABLE 3.1 Number of Models After Applying Each Calibration Criteria

Criteria	Noon NB	Noon SB	Evening NB	Evening SB
Initial	1000	1000	1000	1000
Saturation Flow	159	159	159	159
Statistical Evaluation - WSM	44	93	35	2
Statistical Evaluation - KS	51	101	35	2
Statistical Evaluation - HFF	91	122	100	158
Models Which Satisfied Selected Criteria	44	92	34	2

TABLE 3.2 Number of Common Parameter Sets for Each Approach and Time Period

	Noon NB	Noon SB	Evening NB	Evening SB
Noon NB	44	43	29	1
Noon SB	43	92	31	2
Evening NB	29	31	34	2
Evening SB	1	2	2	2
Common to all Periods and Directions			1	

Given the various sets of parameter values that produced calibrated simulation models, the next step is deciding which parameter set(s) should be used for real world applications. Two types of objectives are presented to aid in this decision.

The first objective is associated with planning purposes, which typically involves the evaluation and communication of corridor level traffic information, such as average travel time and volume. If the intention of the model is to provide general information regarding the traffic's current and future conditions, especially as it relates to the effects of certain strategies or policies, or changes in demand, any of the calibrated parameter sets may be used. In the case of the NGSIM study corridor, any one of the 95 parameter sets may be used as they were deemed adequately calibrated for at least one period and direction. This recommendation does allow for different time periods to utilize different parameter sets. However, where a parameter set(s) is found to satisfy all time periods and directions, one as in this case, it is recommend that this parameter set be included in the analysis.

To obtain a more comprehensive view of the traffic's performance, it is reasonable to perform a study where not only the random seed is altered for each run but the parameter set may also be changed. In this instance, for each enhanced model run, a parameter set is randomly selected from the family of calibrated sets. This approach parallels, and improves upon, a multi-run simulation analysis. Instead of only altering the random seed of a calibrated model, the values of the entire set of effective calibration parameters are changed. Such a change will provide greater insight into the variability of the simulated output. As a precautionary measure, it is recommended that throughout this analysis, the modeler should also monitor saturation flow measures in addition to the other performance measures, to ensure that a simulation's output remains valid.

The second category of objectives is more temporal in nature. This refers to models that are created to communicate corridor and individual vehicle level traffic information in a time-base or real-time fashion. For such purposes, it is recommended to use the calibrated models that best reflect traffic conditions during the analysis period. If traffic information about the Noon period is requested, any number of the 136 parameter sets should be used to deliver the relevant information. Again, using more than one of these parameter sets is encouraged, making the information more robust.

Given the above temporal objective, one of the more immediate expansions of the effort is to have an array of time-of-day calibrated parameter sets which are intended to provide time-based traffic information for periods in which driver behavior maybe be deemed homogeneous. The use of these time-of-day based parameter sets is analogous to applying different signal timing plans, throughout the day, for an intersection. It is expected that there will be multiple calibrated parameter sets for each period. And similar to previous scenarios, these parameter sets should be used as a part of a multi-run simulation analysis – providing a comprehensive view of anticipated performance measures. A further expansion of the time-of-day parameter set construct is the development of a real-time calibration algorithm. The goal of such an algorithm is to allow a real-time data driven simulation model to adapt to changes beyond those of the physical transportation network. This algorithm will be responsible for intelligently adjusting the effective calibration parameters in order for the simulated environment to continue accurately estimating measures – even as drivers' behavior and environment changes.

3.6 Conclusion

Transportation impacts every aspect of daily life. For many decades efforts to improve transportation and alleviate congestion have been made and traffic simulation has played a significant role in these efforts. In light of the growing prominence of traffic simulation, additional emphasis is needed on the calibration of the associated models. Toward this end, a statistical calibration procedure for data-driven traffic simulation model was proposed. This procedure was applied to calibrate the VISSIM model of the NGSIM study area. One thousand potential parameter sets were generated and these parameter set simulations were evaluated against a robust set of calibration criteria to determine which were calibrated. Two calibration criteria were applied: 1) evaluation of startup and saturation flow and 2) statistical evaluation of travel time distributions. The parameter sets that satisfy both these criteria are considered as adequately calibrated. As this procedure requires disaggregate individual data for calibration its large-scale implementation remains challenging. However, with the continuing expansion of individual vehicle based data collection technologies (e.g. GPS probe vehicles, video-based techniques, Bluetooth, etc), it is anticipated that disaggregate level vehicle data (similar to the NGSIM data) will be readily obtainable in near future.

The Monte Carlo based procedure produced 1000 unique candidate parameter sets to calibrate a VISSIM model. To determine which of the 1000 models were well calibrated, a two-part criteria process was applied to the models' output. There were 93 and 34 calibrated models belonging to Noon and Evening time periods respectively. This discrepancy in the number of calibrated models is one of the first indicators that different periods do require different set of values for pertinent calibration parameters. In light of these results it is recommended that for real-time or time-dependent simulation efforts, a corresponding calibration procedure ought to be developed and implemented to bolster the inferences that may be made via simulations' output, even with changing time-periods.

4 Predicting Vehicle Trajectories in Real-Time

The preceding two chapters discussed predictive distributed simulation methods and data-driven calibration techniques. We now shift our attention to predicting vehicle routes which are needed by the simulations. Toward this end we propose a data structure that stores previously observed vehicle paths in a given area in order to predict the forward trajectory of an observed vehicle at any stage. Incomplete vehicle trajectories are conditioned against in a Past Tree, to predict future trajectories in another tree structure - a Future Tree. Many use cases in transportation simulation benefit from higher validity by considering historical paths in determining how to route vehicle entities. Instead of assigning static and independent turn probabilities at intersections, the storage and retrieval of historical path information can give a more accurate picture of future traffic trends and enhance the capabilities of real-time simulations to, say, inform mobile phone users of expected traffic jams along certain segments, direct the search efforts of law enforcement personnel, or allow more effective synchronization of traffic signals.

4.1 Introduction

Computer simulation of road networks and transportation entities is a valuable tool for managing traffic and designing new construction projects. Some transportation models are small in scale and make many simplifying assumptions; others can be very sophisticated and computationally demanding. Many dynamic factors are present in the real world that are difficult to model and predict. For example, say, a particular event takes place that attracts one specific demographic; a simulation model can be made more accurate if it takes into account recently observed trajectories, rather than static turn frequencies at decision nodes. One way to view this information is via decision trees holding relative frequencies at each branching point. Observed trajectories can be stored in a tree data structure (Past Tree) where nodes represent past trajectories up to the given decision point and hold references to other trees (Future Trees) that store observed frequencies of the remaining trajectory. If the target entity is currently in an area where trajectories have been observed in the past, population path similarity can be used to predict where a vehicle will turn or to generate realistic trajectories. For reasons that will become obvious in Section 4.2, we will subsequently refer to Past and Future Trees as simply Past Trees. We will also sometimes use the expression Method of Past Trees (or simply, Past Trees in the charts of Section 4.3) to specifically refer to the querying operation on Past Trees that outputs location estimates (see Section 4.2).

Perhaps the most interesting use of Past Trees is found in the arena of dynamic data-driven application systems - DDDAS (see Darema 2004). Modeling individual vehicles accurately creates a variety of applications. The analysis of emergent, macro-scale phenomena in real-time data-driven types of simulations can be of great interest to various stakeholders. To optimize systems for efficiency, for example, it is possible to model traffic congestion (Fujimoto et al. 2007) and evacuation models (Chaturvedi et al. 2006) with a high degree of validity. The data centers of web search providers have clustered equipment based on the expected workload in different domains (Marin et al. 2013). Past Trees can model consumer preferences and assist in parameter tuning (Ye et al. 2008) and bottleneck analysis of computer networks. Past Trees may also be used in online simulations to determine an accident from cell phone data (Madey, Szabo, and Barabási 2006; Madey et al. 2007) or in law enforcement applications (Fujimoto et al. 2014).

Problem Description

Consider a directed road network G(V, E) where vertices represent intersections and arcs represent road segments. Suppose a transportation entity, such as an automobile, E_T, has been observed at position s_0 on arc e_0 at time t_0 (s_0 is a real scalar in [0, 1] to represent the position *within* the directed arc e_0). Our objective is to estimate the location of E_T at time $t_0 + \Delta t$.

It is helpful, but not necessary, to assume that before t_0, we have observed some partial trajectory of E_T. As an application within the context of dynamic data-driven simulations, we discuss in Fujimoto et al. (2014) an on-line framework for vehicle tracking that may use the Past Trees to iteratively simulate future locations of E_T. The estimated locations are then published to mobile sensors (or a crowd-sourced participant set) which can, in turn, respond with the realized location.

Related Work

Destination estimation is an active field of research. The constraint on the models selected here, however, is that they must be at a conceptual scope that can be mapped directly to a general transportation simulation model. Prediction models for a highly domain-specific area - like handoffs of a mobile phone from antenna-to-antenna (Cheng, Jain, and Van Den Berg 2003), or where the target is at an unknown position, but stationary (Sinclair, Prazenica, and Jeffcoat 2008), for example - are inappropriate within general simulation frameworks. Those that extrapolate from direct vehicular motions (e.g., dead reckoning models; see Cai, Lee, and Chen 1999) are too granular and impose an unrealistic behavior on a given vehicle entity. We also ignore models that depend on the identification of a particular vehicle and its historical data (Laasonen 2005).

In the next section, we summarize a prediction model based on efficient routes (Krumm 2006). The model predicts based on the idea that drivers tend to make choices that are fairly efficient when attempting to reach their destination. Unlike Past Trees, Krumm's prediction model does not use any historical trajectory data. Nevertheless, we will use it as a benchmark in Section 4.3.

Krumm's Prediction Model

John Krumm developed a static prediction model that assigns higher likelihoods to areas that are consistent with the path taken thus far. Thus, inefficient potential forward trajectories are given less weight than those that are consistent with efficient behavior. It is important to note that this model does not utilize previously recorded trajectories, outside the partial trajectory of the vehicle currently being observed. Krumm assigns each cell, c_i, in a region an estimated probability, based on the observed path, S, taken thus far, using Bayes' Law:

$$p(c_i|S) = \frac{p(S|c_i) \times p(c_i)}{\sum_{j=1}^{N_c} p(S|c_i) \times p(c_i)}.$$

Krumm has empirically determined that the probability of an efficient cell transition is 0.625 in a particular experiment in Seattle, WA. For a particular cell ci in the grid, each cell of the observed trajectory is multiplied by 0.625 if it brings the target closer to ci, and by 1-0.625 if not. This is then multiplied by an overall cell bias term $p(c_i)$ and normalized.

4.2 Past and Future Trees

Conceptual Structure and Trajectory Insertions

In order to make proper use of Past and Future Trees, a number of population trajectories must be sampled first. In the following conceptual description of the structures, we assume that this data has been collected already and has been inserted into the structures. A discussion on trajectory insertion (see Figure 4.1) follows this description. Querying Past Trees to make predictions (i.e., the Method of Past Trees) is discussed in the next section.

Every decision point in a region (e.g., the road on the western side on a particular intersection) is associated with an instance of a Past Tree and each node in a Past Tree links to an instance of a Future Tree (see Figure 4.1). The vertices of a Future Tree contain traversal counts as well as destination counts. If a partial trajectory, $T_p=(e_0, e_1,...,e_f)$, of a vehicle up to a decision point has been observed, the corresponding Past Tree is traversed in the reverse order of T_p (the root corresponds to e_f). Traversing the Past Tree in this fashion is equivalent to conditioning against the observed trajectory. This means that if no partial trajectory up to e_f, had been observed, we would simply access the Future Tree associated with the Past Tree's root node. If the currently observed partial trajectory does not exist in the tree, the traversal only goes down to the node that does not have the requested child. Once the corresponding Future Tree is discovered, it is traversed to yield the measure of interest. Some of these metrics are discussed in the next two subsections. For example, if one wishes to construct an estimated destination probability map, a breadth-first search for positive destination counts can construct a frequency distribution, leading to the desired map.

In Figure 4.1 below, we assume no previous trajectories have been observed (i.e., this is the first trajectory being observed). Further, this trajectory will cause updates on every Past Tree adjacent to the realized path. However, for illustrative purposes, we will only focus on the updates to the Past Tree corresponding to the western side of Intersection 3, PastTree3, W. A vehicle has approached Intersection 3 by first traversing Intersection 1 and then Intersection 2. This partial trajectory is inserted into PastTree3, W (the solid black arcs). After that point, the remaining path is inserted in the Future Trees of all vertices in the path up to Intersection 3 (the lightly dotted arcs), so that any future read request on the Past Tree may access this particular observation for any partial trajectory that follows the same pattern up to the current decision point (irrespective of the length). The Future Tree's traversal counts are incremented if the vehicle traverses the given intersection and the destination counts are incremented if the vehicle stops in the given intersection's proximity (i.e., terminal point along the trajectory). As mentioned before, we assume this is the first trajectory being observed. Had the Past Tree already been partially populated, it could have been the case that no additional arcs would have had to be inserted. However, the counts would still need to be incremented.

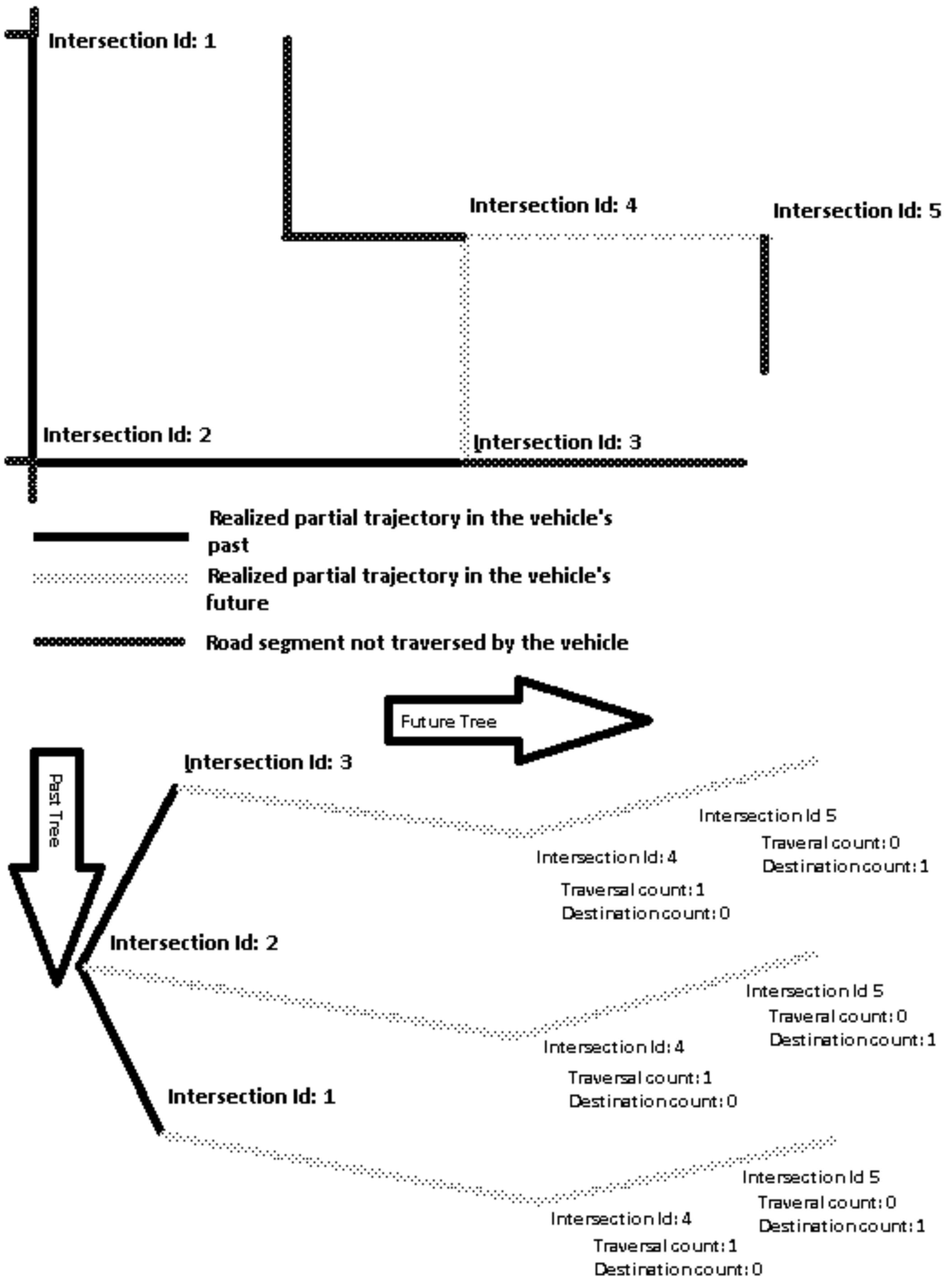


Figure 4.1: Past & Future Tree visualization

The Method of Past Trees: Predicting Trajectories and Destinations

Once a Past Tree for a given decision point is populated, it can be used to make predictions for a vehicle approaching the given decision point. If the vehicle's trajectory up to the point has been observed, the given Past Tree can be queried to retrieve the corresponding Future Tree. Predictions regarding any future decision point can be made by considering the corresponding nodes at that depth of the Future Tree. It should be noted that this yields the nth decision into the future and, therefore, has limited use: These points could be asynchronous in the expected arrival time (e.g., the example travel times to the Future Tree nodes in Figure 4.2). A prediction for Δt time units into the future requires locations to be tagged with point estimates of the travel time it takes to reach the respective points. A general search algorithm can then return the set of feasibly reachable locations (the intersection of the Future Tree and the shaded line cut in Figure 4.2). The frequency distribution of the traversal counts in this set can then be used to build an estimated probability mass on the map. The following pseudocode (for a parallel shared-memory machine), applied on the Future Tree of an observed vehicle, yields this prediction (a probability map):

```
Input: Future Tree (this instance), Δt
1 i:=0
2 root.travelTime := 0
3 while (there is at least one non-leaf vertex of travelTime < Δt at depth $D_i$) {
4   Par for all ( non-leaf vertices u ∈ $D_i$ with u.travelTime < Δt ) {
5       Par for all w that are adjacent to u {
6           <edgeTravelTime, positionAtΔt> := simulateTravel((u,w,Δt))
7           w.travelTime := u.travelTime +edgeTravelTime
8           if (w.travelTime > Δt) {
9                   w.late := true
10                  frequencyMap.populateCountAtPosition(positionAtΔt, u, w)
11          }
12      }
13  }
14  sync()
15  i := i+1 //master process only
16 }
17 probabilityMap := frequencyMap.transformToProbabilities()
18 return probabilityMap
```

In general, this depth-synchronous implementation is not efficient, even if the underlying work-scheduler balances optimally: A time-parallel simulation (Kiesling and Luthi 2005) may simultaneously evaluate delays across road segments on differing depths and, therefore, offers the flexibility to saturate all the available processors with work. Nevertheless, we focus on this implementation for ease of exposition. The procedure expands all simple paths from the root node (Lines 3-4) until all the terminal vertices are tagged with a travel time greater than Δt; at this point, the potential locations of the vehicle are found. Pursuant to this goal, vertices are tagged with a point estimate of the travel time to reach the respective intersections by invoking the transportation simulation, simulateTravel, on Line 6; this subroutine returns the travel time on the road segment from vertex u to vertex w. Unless this subroutine is implemented in a trivial fashion (for example, by simply retrieving a point estimate from historical data and not considering live data), the execution time of this call will dominate the total runtime of the program. (The time used by this bottleneck primarily depends on the granularity of the simulation model; a computationally efficient microscopic traffic simulation model can be found in Nagel and Schreckenberg 1992.) If the cumulative travel time up to w is greater than Δt, simulateTravel will also return the predicted position of the vehicle at Δt (positionAtΔt). At this point, the observed traversal count is queried from the corresponding Future Tree (from vertex w is the vehicle is closer to w than to u; otherwise the traversal count from u is used). This count is inserted into a map representing the frequency distribution at Δt (Line 10). After this map is populated, it is converted to a probability mass on the map and returned (Lines 17-18).

If the question is what destination the vehicle is likely to approach, the destination counts described in Section 4.2 are fetched with the help of any tree search procedure (e.g., breadth-first search). The non-zero destination count of every node in the future tree is inserted into the appropriate location in the frequency map, which is subsequently transformed into a probability map and returned.

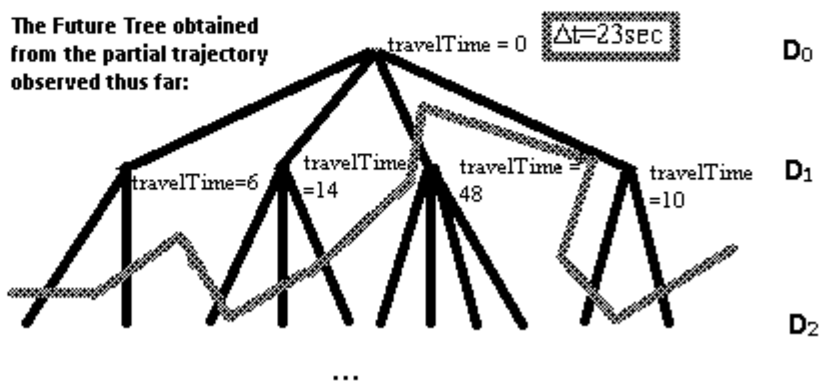


Figure 4.2: The Future Tree is used to retrieve the potential locations of the vehicle Δt into the future. In this case, we identify all vertices that are reached from the current location within a time span of 23 seconds. The shaded line illustrates the possible locations at that time.

Generating Simulation Trajectories

Suppose we have constructed a subgraph that represents some road-connected intersections and now we wish to generate and route vehicle entities within. The following pseudocode illustrates these steps at a high level:

```
Input: Future Tree (assume code is applied to this instance), vehicle entity e
1 totalCount := 0
2 for all ( vertices u ∈ $D_1$ ) {
3  totalCount := totalCount + u.traversalCount
4  traversalCountBoundaryList.add(totalCount)
5 }
6 $U_0$ := realization from a discrete uniform between 0 and totalCount
7 toNode := computeToNode( traversalCountBoundaryList, $U_0$ )
8 e.setNextNode( toNode )
9 sampleSpaceCardinality := toNode.destinationCount + toNode.traversalCount
10 e.stopAtNextNode := Bernoulli(toNode.destinationCount / sampleSpaceCardinality)
11 return e
```

Vehicle instances will still be created as usual (e.g., from a non-homogeneous Poisson process). Once the entity is in the model and reaches a decision point, the simulation engine will query the past tree corresponding to a decision point for the trajectory taken up to that point. This, then yields a reference to the corresponding Future Tree. At Depth 1 of the tree, the observed frequency counts of all sibling nodes are read and inserted into a list, traversalCountBoundaryList (Lines 2-5). A random number (Line 6) is then transformed to a decision from this empirical frequency distribution via its c.d.f. (Line 7) and assigns it as the next location of the vehicle entity (Line 8). However, the vehicle entity may stop at that point. To account for this, we first realize the next intersection (Lines 1-8) and then condition against it (Lines 9-10). Both the destination and the traversal counts are known, so whether the vehicle stops can be inferred from a Bernoulli realization with success probability equal to the destination count divided by the sum of the destination count and the traversal count of the realized vertex (Line 10).

Space-Efficient Storage

The naive storage mechanism (i.e., a tree for each decision point) has much redundancy, but offers fast access to the future likelihoods (in fact, linear time in the trajectory length taken thus far). One observed trajectory will cause insertions into every tree along its path. This redundancy causes much wastage of storage. More precisely, suppose the length of the typical trajectory is n and the total number of observed trajectories is T. This vector of size O(n) is pushed into n Past Trees, but in each Past Tree it is replicated n times because it is recorded for each partial trajectory up to the point represented by the tree. The total storage requirement in this representation is thus O(T × n3).

A sensible approach would be to cut off the tree at a certain depth - one should expect diminishing returns on accuracy when going down the Past Tree at significant depths (this is obviously not true in all practical situations). The results in Section 4.3 truncate the Past Trees at Depth 3.

A better approach, reducing the storage requirement from O(T × n^3) to O(T × n), is to use an array indexed by directed intersection identifiers (this is with almost no loss of generality as a different identification scheme can be coupled with a hashing rule with no performance loss in the average case). Each cell in the array will hold a reference to the head of a linked list. Such a linked list will simply contain pointers to trajectories containing the corresponding directed intersection identifiers (see Figure 4.3). Every time a trajectory is observed, all linked lists that contain a directed intersection of that trajectory will be augmented by one more reference. Thus, the storage requirement and the total insertion costs are limited to O(T × n). The potential for a highly skewed length distribution (on the collection of trajectories on directed intersection identifiers) motivates the use of lists, rather than arrays. Once a prediction is requested (when a vehicle has entered a particular directed intersection), a significant number, k, of previously observed trajectories are queried: The corresponding header can be accessed in constant time (random access with a specific directed intersection id) which allows for the extraction of the k trajectories of interest. Building a corresponding tree (by parsing and inserting the k trajectories) takes linear time in the size of the trajectories if partial trajectories outside of the currently observed one are omitted, leading to a total cost of O(k × n).

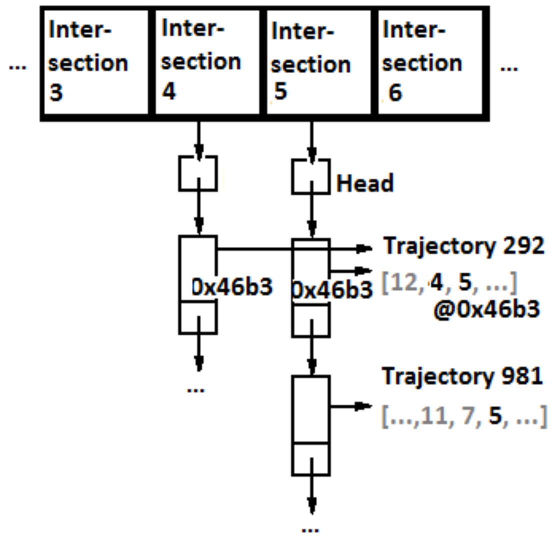


Figure 4.3: Illustration of the space-efficient storage scheme. Trajectory 292 is stored (only once) at location 0x46b3. Since it contains Intersection 4 and 5, the lists of these two intersections contain pointers to the trajectory's memory location.

4.3 Experimental Evaluation

To evaluate the accuracy of Past Trees, separate experiments were performed on randomly generated input as well as real-world data. Both experiments are run on cellular automata (CA) and generate two different error plots each (lower quantities are more desirable). In both cases, the directed graph representation introduced in Section 4.1 can be mapped in one-to-one correspondence with the CA if the appropriate cell transition rules are added.

The Method of Past Trees is compared to the method described earlier (Krumm 2006), which does not require previously collected trajectories. Whether the data acquisition and storage costs for Past Trees are acceptable depends on the added value from the increased accuracy for a given application. These costs scale with the desired sample size and the size of the target area.

Synthetic Experiment: Setup

A computational experiment was performed to sample the degree of accuracy provided by Past Trees. Here, the input (i.e., the trajectories of the drivers) is generated synthetically.

The model is based on a 20-cell-by-20-cell cellular automaton that approximates a two-dimensional metric space; this discretized plane serves as a conceptual model of a real-world region with pervasive road network coverage. 200 trajectories are sampled as follows: The origin and destination are first sampled by the pseudo-random selection of two cells. In this selection, some cells are more likely to be realized than others because cell popularity is assigned - at the beginning of the simulation - from a probability distribution that mimics the population distribution of US cities. Each cell is assigned a probability by first sampling the lognormal distribution with μ=7.28 and σ=1.75 (which is a good fit for the population distribution of the United States - see Eeckhout 2004), evaluating the density at those realizations, and then normalizing these values into a probability mass. Further, positive correlation is forced upon the popularities of neighboring cells (ρ=0.3): Less (more) popular cells are more likely to be near less (more) popular cells.

After the origin and destination are determined, the cells in between them are generated. At each intermediate cell, the vehicle will transition to a cell that is in accordance with the most efficient route with probability 0.625, and to one that is not with probability 1-0.625; we call this behavior the desire for efficiency. Each cell also has a statically assigned 10% chance of being an entry point that leads into a predetermined tunnel of neighboring cells, a preferred path. The length of a preferred path is geometrically distributed with a mean of five cells. A cell that leads into a preferred path is privileged to temporarily override the desire for efficiency until the vehicle exits the preferred path. Preferred paths (and their entry points) are randomly generated at the beginning of the simulation experiment and don't change during the execution. In this scenario, preferred paths have a mean length of five cells. The rationale of including preferred paths is to model unexplained trends within the population. Of the 200 simulated trajectories, 80% are first used to populate the trees; the remaining 20% are used to predict destinations and compute errors. The depth of Past Trees is limited to three cells.

Synthetic Experiment: Results

Figure 4.4 shows the mean accuracy as a function of the fraction of the trip completed (ensemble averages across the 0.2*200=40 trajectories). At each transition of any trajectory in the test set, the error model queries the Past and Future Trees of the current cell for all potential destinations of the vehicle. It then computes the rectilinear distance from the vehicle's current position to the center of the estimated probability mass (the predicted destination) as well as the rectilinear distance from the vehicle's current position to the actual destination. The error in Figure 4.4 is the absolute value of the difference of these two values.

Across all the simulated trajectories, for each cell transition, an error is computed and inserted into the bin corresponding to the tenths precision bin of the trip completion fraction. Then, for every completed trip fraction bin, the average error is selected. 0.8*200 = 160 trajectories were used to populate the trees.

As the vehicle entity approaches its destination, more data can be used because the partial trajectory up to the given cell is available: In the case of Krumm, it points towards efficient behavior, and in the case of Past Trees it points towards previously observed trajectories that are similar. This explains the steady improvement to a trip completion of 0.7. The increase of the plot at the end of the trip is a result of the vehicle approaching its destination relatively quickly and thus decreasing the first term mentioned above (distance to actual destination), while the second term (distance to predicted destination) is exposed to diminishing returns.

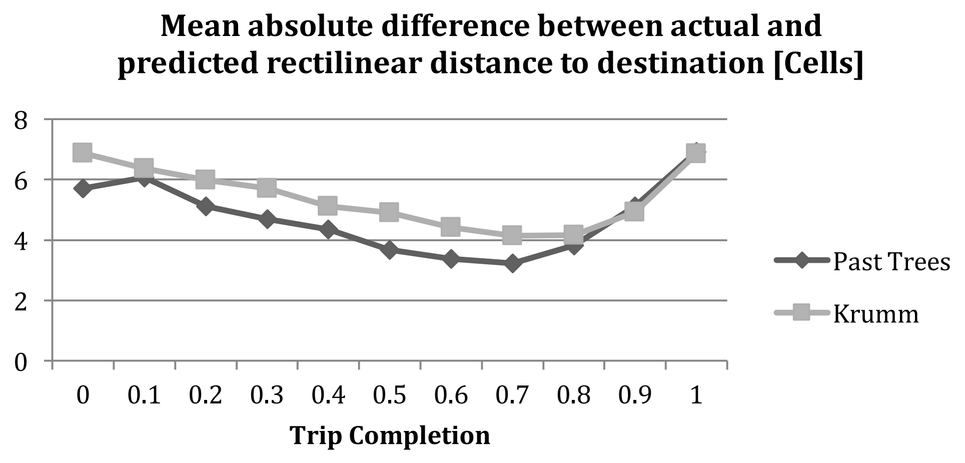


Figure 4.4: Error plots of the mean absolute difference between actual and predicted rectilinear distance to destination

Figure 4.5 simply shows the rectilinear distance between the predicted and actual destination. This quantity hovers around 10 cells for both prediction models (with a slight improvement towards the end of the trip). This is reasonably close in a 20-by-20 grid; no miracles are to be expected in a scenario where the trajectories of vehicles almost exhibit the patterns of 2D random walks with cell transitions that are independent of one another.

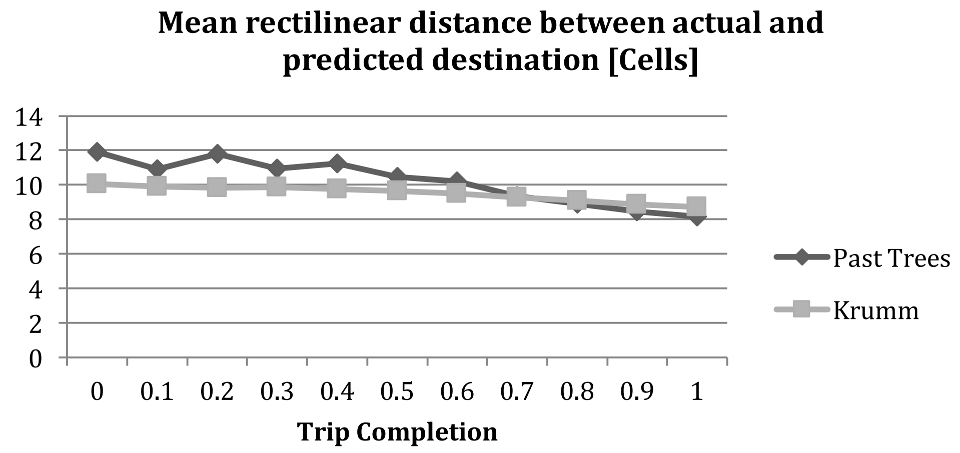


Figure 4.5: Error plots of the mean rectilinear distance between actual and predicted destination

Real-World Traces: Setup and Results

A deterministic simulation was performed using the detailed NGSIM Peachtree Street dataset, collected between 4:00p and 4:15p on November 8th, 2006 in Atlanta, GA (NGSIM Community Home 2014).

The road segment was modeled as a 30-by-3 cellular automaton and the first 80% of the 461 observed trajectories were used to populate the trees (the remaining 20% were used to test the model).

The following plot illustrates the error computed in the same fashion as illustrated in Section 3.1.2 for Figure 4.4. It should be noted here that Krumm's model is not well-suited for an almost one-dimensional environment as the estimated destination probabilities will just be uniform across the entire road. Because the sample size is small, the first 80% of each trajectory use superimposed Past Trees for each cell.

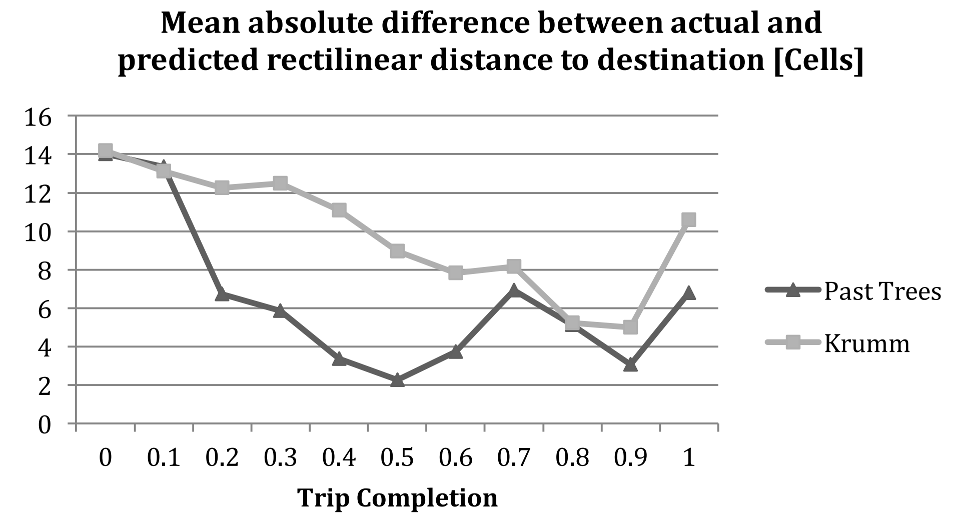


Figure 4.6: Error plots of the mean absolute difference between actual and predicted rectilinear distance to destination

Figure 4.7 presents errors in the same way as Figure 4.5. For the Past Tree errors, the increased values towards the end of a trip are the result of time not being considered in the models: As the vehicle entity is in one of the final cells, the model assumes that the partial trajectory up to the current point is indifferent from one that is near the center of the trajectory, which leads to an estimated cell much farther away. This may be avoided by not only incrementing destination counts, but also noting the progress along the trajectory during the data collection phase.

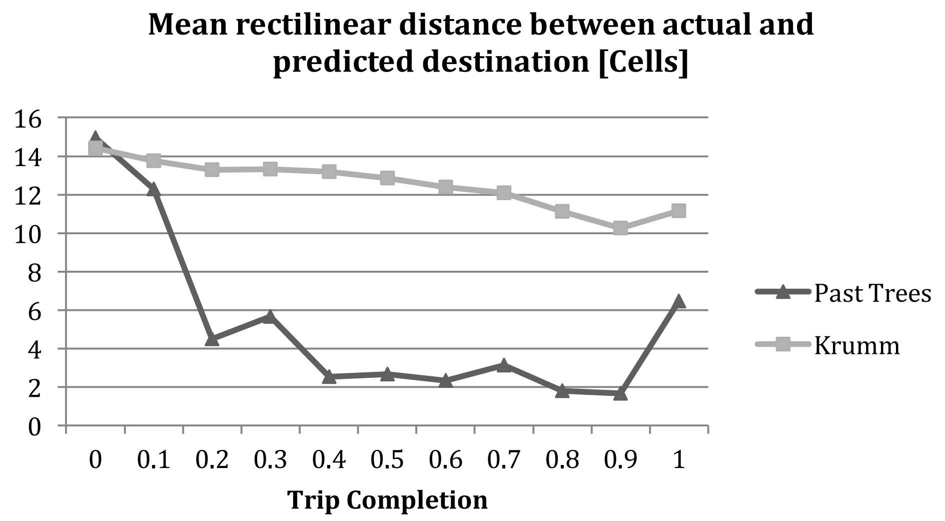


Figure 4.7: Error plots of the mean rectilinear distance between actual and predicted destination

4.4 Conclusion and Future Work

A structure to store trajectories along road segments was introduced. Past Trees can be used to either generate trajectories for simulation entities or to predict likely destination of a vehicle whose path is being observed. The experimental results show that the structure gives a solid prediction framework. Modeling vehicular movement is clearly of great economic value as driver behavior and traffic management can be made more efficient.

In the future, we plan to investigate enhancements to Past Trees, such as conditioning against more factors as well as weighing more recently observed trajectories more heavily; applying statistical regression and exponential smoothing to the Method of Past Trees has the potential to improve accuracy. Although it was mentioned above that every decision point has one past tree associated with it, it can be very useful to have multiple trees to account for factors such as seasonality (the same tree can be used if it is 'colored') or other factors like congestion level. Traffic exhibits seasonal characteristics; for example, drivers might opt for a particular pathway during rush hour in order to avoid congestion in other areas.

It seems reasonable that more recent collections are most heavily correlated with present and near-future trajectories. For example, a popular sports game may tilt historical behavior into a particular direction. If paths are proactively collected in a particular area, exponential smoothing can adapt to recent trends.

5 Efficient Execution of Replicated Transportation Simulations

Many Dynamic Data-Driven Application Systems (DDDAS) use replicated simulations to project possible future system states. In many cases there are substantial similarities among these different replications. In other cases output statistics are independent of certain simulation computations. This chapter explores computational methods to exploit these properties to speed up the simulation execution time. We discuss a new algorithm to speed up the execution of replicated vehicle traffic simulations, where the output statistics of interest focus on one or more attributes such as the trajectory of a certain "target" vehicle. By focusing on correctly reproducing the behavior of the target vehicle and its interaction with other modeled entities across the different replications and modifying the event handling mechanism the execution time can be reduced. A speculative execution method using a tagging mechanism allows this speedup to occur without loss of accuracy in the output statistics.

5.1 Introduction

There are many applications that utilize microscopic traffic simulation models driven by online data to analyze and optimize operational transportation systems (e.g., (Suh, Hunter, and Fujimoto 2014)). For example, for this chapter we posit a motivating DDDAS application concerned with tracking the location of a vehicle where continuous surveillance is either not possible or not desirable. Such a DDDAS deployment may repeatedly execute a cycle that includes (1) detecting the current location of the vehicle, (2) predicting the likelihood of possible locations some time into the future, and (3) focusing surveillance efforts on the most likely future locations, e.g., by re-positioning sensors or concentrating data analysis efforts in certain areas (Fujimoto, et al. 2014). Realizing this DDDAS application may require the completion of many replicated simulation runs. For instance, the second step involves executing a set of replicated simulation runs where each run models the target vehicle following a different potential route to reach some (potentially unknown) destination. In addition, to further optimize sensor repositioning other unknowns may need to be captured such as changing traffic conditions and travel time variability within a traffic condition. In this chapter we propose to exploit the similarities among these runs to optimize execution time or minimize needed computational resources.

For example, in runs that only differ according to the route taken by the target vehicle the trajectory of vehicles not impacted by the target vehicle will remain the same across the different runs. Further, the output statistics of interest may not require portions of the simulation to be completed at all as simulation computations concerning vehicles far away from the target as the target approaches its destination will not affect statistics related to the target vehicle. The focus of this chapter is to develop and evaluate techniques to reduce the amount of computation required given these similarities, thereby reducing the time required to gain future estimates and/or reduce the computational resources that are required for the DDDAS application.

Here we present an algorithm to accelerate the generation of results from *N* similar, replicated simulation runs without loss of accuracy. In other words, the statistical results that are produced should be identical to those produced by a brute-force approach of completing N independent simulation runs. The problem is stated as follows. Let E_T denote the target vehicle, i.e., the vehicle being tracked. Let us assume its position is known at time t_0 and that a prediction model has determined a set of possible future paths and their likelihoods. The simulation now aims to determine the time at which E_T reaches certain user-specified points along each of these paths. After extracting real-time information such as current traffic conditions throughout the region, the traditional approach would perform *N* replications each simulating E_T using a different route. Our goal is to minimize the amount of computation that must be performed while still computing the same results as the traditional replicated approach. An extension of this work is to consider multiple replications using a single route, e.g., to consider the impact of stochastic or other variations, however, we will not explore this direction here.

5.2 Related Work

An application such as this requires one to predict a set of likely destinations for the target vehicle. There have been numerous efforts that have developed methods to predict this destination set. For example, in (Pecher, Hunter, and Fujimoto 2014) the routes the vehicle might take and aspects such as traffic congestion that will impact its travel time are considered. A prediction model that assigns destinations a higher likelihood if they are consistent with the path taken thus far can be found in (Krumm, 2006). Neural networks have also been investigated for use in destination prediction (Miklušćák et al., 2012). If limited data are available, the SubSyn algorithm from (Xue et al., 2013) can be used to synthesize new trajectories from previously collected ones. A summary of relevant models can be found in (Pournajaf, Xiong and Sunderam, 2014). Given these and other efforts the purpose of this chapter is not to modify a previous prediction method. Rather, the aim is to illustrate how one can *quickly* execute a traffic simulation that projects the most probable positions of the vehicle at some point in time into the future.

Sharing computation among replicated simulation runs in the context of cloning running simulations is described in (Hybinette and Fujimoto, 2002), which served as motivation and the starting point for this work. The approach described here differs in three ways. First, the approach described here focuses on transportation simulations though many of the ideas can be generalized to other applications. Second, we introduce speculative execution as a means to improve performance. This involves considering the output statistics produced by the simulation in order to determine those simulation computations that must be completed. Third, like the approach described in (Hybinette and Fujimoto, 2002) portions of the simulation are replicated as needed during the computation as the different replications diverge using a technique called incremental cloning. Here, incremental cloning is applied to selected state variables as needed rather than an entire simulation process, yielding a more efficient implementation.

Another related approach is called updateable simulations (Ferenci et al., 2002). This is a technique that first executes a baseline simulation run, creates a log of the entire execution, and then computes the results for other replications by determining the computations that are different and updating the log, thereby reusing results from the baseline run. The updateable simulation approach analyzes the events of the baseline simulation for potential speedups in the management of the Future Event List (FEL) for subsequent runs. The approach described here does not require completion of a baseline run and the creation of an event history log, nor the management of an FEL. Further, for each different scenario, the updatable simulation algorithm must explicitly execute another run (albeit with reduced computation), which is avoided with the algorithm presented here. Lastly, the reuse discussed here does not stem from unchanged events, but rather from the invariance in certain state variables across the different scenarios.

The approach described here uses the concept of minimum propagation delays, i.e., the minimum amount of simulation time that must elapse before one object in the simulation can affect another. This is a familiar one in the parallel discrete event simulation literature. Stemming from its origins captured in a term called lookahead (Chandy and Misra, 1979), this concept is generalized as the distance between objects and used to synchronize parallel simulations, e.g., see the work described in (Ayani, 1988), among others. Here, distance between objects is exploited to help determine those computations that may affect the statistics produced by the simulation in order to improve the efficiency of the proposed algorithm, especially the tagging mechanism described later. For example, if statistics are needed for the target vehicle E_T at simulation time T, one can use the distance between objects concept to determine those vehicles that might affect the statistics concerning E_T at time T.

5.3 Simulation Algorithm

For this study we use the Nagel & Schreckenberg (1992) model for traffic based on cellular automata. Although the concepts presented here generalize to other temporal and spatial resolutions, it is straightforward to illustrate the concepts used in the simulation algorithm with the Nagel-Schreckenberg model. In this model, a road is mapped to an array of neighboring cells. Each cell either contains a vehicle or is empty. A vehicle, at any given time, includes a speed attribute that is represented as an integer. A global maximum speed is also defined. At each time step, the following actions are applied to every cell-occupying vehicle:

1. The speed of the vehicle is incremented if the vehicle is not at the maximum speed.
2. The current vehicle speed is compared to the number of empty cells in front of the vehicle. If the number of empty cells is smaller, the speed is reduced to the empty cell count (in order to avoid a potential collision).
3. If the vehicle's speed is positive, it is decremented with probability p (this is a predefined parameter).
4. The vehicle advances forward by the number of cells equal to the corresponding speed.

Since 1992, multi-lane models have been proposed where it is possible for vehicles to overtake each other. This aspect is not modeled here, however, the algorithm proposed below easily generalizes to multi-lane models.

The approach used here is summarized as follows. The source of variation among the replications stems from different routes taken by the target vehicle E_T. To simplify the discussion we assume the statistics of interest are concerned with properties of E_T, e.g., its temporal-spatial trajectory through the road network over time. We superimpose N computational sequences, each modeling one trajectory of E_T onto a single run where E_T is absent. Each such trajectory is produced by what is referred to as a virtual instance of E_T, i.e., a virtual vehicle (VV). The other non-target vehicles in the simulation are called model vehicles (MV's). Whereas the traditional approach simulates one instance of E_T and all of the MVs for each of the N replications, here the *single* superimposed replication simulates N VVs, and initially one set of MVs. Each VV is therefore logically associated with one replication in the traditional, brute force approach.

If an MV interacts with a VV and as a result behaves differently compared to the case where E_T is absent, the simulated trajectory of MV is erroneous; for example, MV may need to reduce its speed to account for the VV. In this case the MV is referred to as a *hazardous* MV. A hazardous MV is at risk of causing errors in the output statistics, however, a hazardous MV may not actually affect these statistics. For example, if the correct behavior is the MV should slow down, its corrected behavior may not have any impact on the trajectory of VV, which is the focus of the simulation. Additional MVs that interact with a hazardous MV may also become hazardous. In order to determine if the output statistics have been compromised, a tagging mechanism is used to flag "hazardous" MV's and their actions. Once a hazard is confirmed to have impacted the output statistics, a new execution path is created (a physical "clone," using the terminology from (Hybinette and Fujimoto, 2001)) that explicitly computes the events for the specific VV that originally caused its first tag to appear. During certain timesteps, the simulation may clear tags that are guaranteed not to cause any errors in the output statistics. It should also be noted that in most cases, *one* superimposed replication may not capture sufficient variation in the environment. In order to capture not only variation among the attributes of E_T, but also the simulation environment, the modeler may use a batching scheme for a set of superimposed replications. Section 4 shows how many VV's should be realized within a single superimposed replication for a particular case study.

Superimposed Execution Before Hazards Arise

In the following illustration, we adapt the original Nagel-Schreckenberg model for our purposes. We assume that the parameter p is only global for MV's. Each E_T instance realizes its own p_i parameter from a predefined density before it spawns. Further, each cell may contain any number of vehicles in the proposed approach, but there may at most be one MV contained within one cell (the reason for this will be explained shortly). Each vehicle samples a routing sequence from a probability table before it spawns and owns this sequence throughout its lifetime. All vehicles (MV's and VV's) will execute all four steps outlined in the previous section during each time step. In the examples below, we assume that the output statistics of interest are limited to measurements of the target entity E_T, namely the time taken to reach the final point of the routing sequence.

The left part of Figure 5.1 illustrates the traditional replicated approach executing the model. Each array of cells is a subsection of the road in a given replication at timestep 44. The speed reduction probability attribute p_i of each E_T instance is depicted; the routing sequence attribute, which governs the vehicle's behavior at intersections, is not displayed. One instance of the target vehicle, E_T (the cells with light orange shading), is shown for each replication. The model vehicle (light grey shading) in the rightmost cell has not been influenced by E_T in all three replications. It is important to note that the behavior of the environment does not change across replications, unless the environment is influenced by E_T. Note that the model vehicle shown in the leftmost cell in replications 2 and 3 has reduced its speed in Replication 1, prior to Timestep 44, due to the presence of a rather slow instance of E_T. Thus, the leftmost cell in replication 1 is empty while it is occupied in replications 2 and 3.

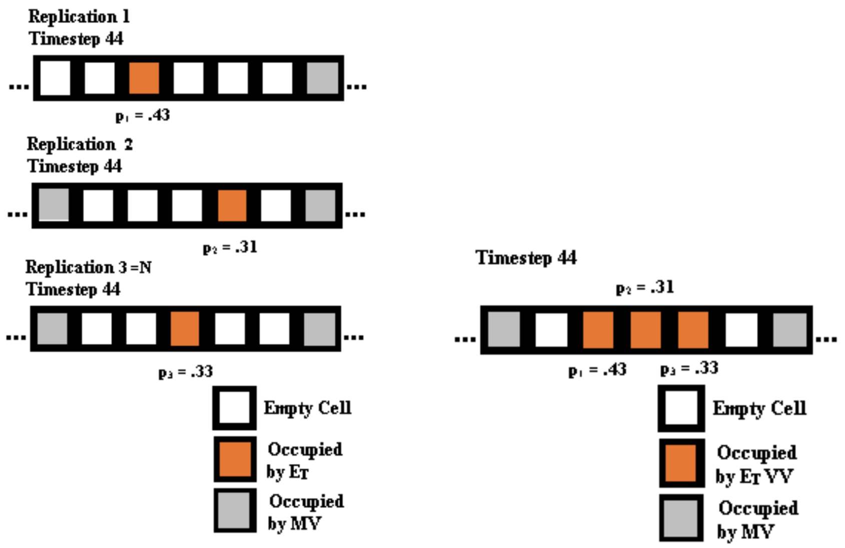


Figure 5.1: The traditional execution approach (left) and the proposed approach (right)

The right part of Figure 5.1 shows a snapshot of the simulation where the three replications are superimposed into one execution. Note that all three VVs appear in the superimposed execution. During collision detection (Step 2 of the Nagel-Schreckenberg algorithm) all vehicles will react to MV, but MVs will ignore virtual vehicles. This implies that it is possible for a cell to now contain both zero or one MV along with any number of VV's. Virtual vehicles do not interact with other virtual vehicles because they logically reside in different replications. VVs do interact with

MVs, however. Lastly, MVs react to other MVs as in the traditional approach. The fact that they ignore VV's can create hazards. For example, the leftmost cell of the superimposed simulation is shown to contain a vehicle, even though that cell is empty in replicated simulation 1 (shown on the left). This vehicle represents a hazard for replication 1 in the superimposed simulation. In the following we refer to this vehicle as model vehicle A.

Speculative Execution with Hazards

As noted above, in replication 1 shown in left part of Figure 5.1 MV *A* was missing from the leftmost cell at Timestep 44, but is present in the superimposed simulation as well as the other two replications; the superimposed simulation represents the location of *A* in the absence of the virtual vehicle. This invalid state does not directly pose a problem because the desired output statistics are concerned with the virtual vehicle, and the virtual vehicle's behavior is not affected by *A*. However, it is possible A does affect another model vehicle *B*, e.g., a fast moving vehicle that overtakes *A* and then interacts with the *VV*. Unless precautions are taken the *VV*'s output statistic may be incorrect if the behavior of B is not taken into account in the superimposed simulation. To address this problem, *A* is flagged as a hazard. When B interacts with *A* it too is also flagged as a hazard, and when B interacts with the VV the error is detected, and a corrective action, termed a rollback, must be executed.

The approach used here is termed *speculative execution* because VV's behavior is modeled assuming VV is oblivious to the impact that VV has on the rest of the simulation. This will not impact the results of the simulation so long as VV's impact on the rest of the simulation do not affect VV itself. If this assertion turns out to be incorrect, as exemplified by the above example, corrective actions are required to ensure the output statistics produced by the simulation match the brute-force replicated approach.

Once a model vehicle alters its behavior because of a virtual vehicle in front of it, there are now two states to consider:

- *State A*. The model vehicle does not consider the virtual vehicle and checks if there is another vehicle present in the road segment it tries to traverse in the current timestep. This is the valid state for all virtual vehicles except the one that was ignored.
- *State B*. The model vehicle considers the virtual vehicle during the collision detection step. This is an invalid state for all virtual vehicles except the one that was considered.

Because there are more virtual vehicles beyond the one being considered, it is beneficial to simulate using state A. In order to confirm whether this causes an error for the virtual vehicle that was ignored by the model vehicle, two possible sources of error must be considered:

- Error-Source 1. Had the model vehicle been blocked by the virtual vehicle, it could have caused model vehicles behind it (upstream) to slow down. In response, those model vehicles could alter their behavior (routing sequence, aggression etc.) and impact the virtual vehicle at some point in the future.
- Error-Source 2. The model vehicle behavior that ignores the virtual vehicle can also cause errors. While the virtual vehicle can ignore the specific model vehicle that passed through it, the model vehicle can now influence the behavior of other model vehicles in front of it that can impact the passed virtual vehicle.

In either of these situations two error-causing events may occur:

- Error I. The virtual vehicle encounters a model vehicle that it should not have encountered.
- Error II. The virtual vehicle does not encounter a model vehicle that it should have encountered.

If a model makes Error-Source 1 possible, one can simply clone the model vehicle and consequently run the scenario of the virtual vehicle along with the superimposed scenario, rather than to use the tagging rules (which are discussed shortly) for both the passing model vehicle *and* an alternative (blocked) instance of the same model vehicle. Which mechanism is more efficient (explicit cloning vs. two tagging sources) depends on the specific application. If the traffic intensity is high, it is likely that explicitly cloning all vehicles could be computationally intensive. At the same time a high traffic intensity is likely to trigger a rollback when the tagging mechanism is used.

If only Error-Source 2 is possible, a forward tagging mechanism may alternatively be used to detect errors (explicit cloning could still be the better option for certain models). As soon as a model vehicle overlaps or overtakes a virtual vehicle, a tag is added to the model vehicle. A tag contains the *id* of the virtual vehicle, an occurrence count, and a flag that specifies whether this is the first model vehicle affected (this would be set to *true* as this is the first time the particular occurrence *count-id* combination has been observed). Whenever another model entity slows down - in the collision detection step - as a result of a tagged model entity in front of it, the tag is copied (with the flag field now being *false*). Additionally, when a tagged model entity reaches an intersection, the tag is added to the intersection. As soon as the virtual vehicle encounters one of its prior tags (at an intersection or in the collision detection step), the simulation explicitly executes the replication with the given virtual vehicle as the sole instance of E_T (as in the traditional approach) and the virtual vehicle along with all its tags are removed from the superimposed model. If other statistics from the virtual vehicle were collected, they must be rolled back. A virtual vehicle that encounters its tag on a model vehicle in front of it implies an Error I, while an encounter with its tag on an intersection could imply an Error II. The following listing shows the full algorithm, which replaces step 2 of the Nagel-Schreckenberg algorithm:

```
collision_detect(vehicle e){
    if (e.type == virtual)
            1. e doesn't  collide with other virtual vehicles
            2. e collides with model vehicle that are not tagged with e's id but...
            3. ... e doesn't collide with model vehicles that have previously illegally overtaken e
    or overlapped with e (check flag if the model vehicle is tagged with e's id) and...
            4. ...if e collides with a tagged (of its own id) model vehicle, remove e from current
    model (along with all its tags) and physically clone e on a separate replication.
    if (e.type == model)
            1. e collides with model vehicles but...
            2. ... if e collides with a tagged model vehicle, e will copy the tag(s)
            3. e doesn't collide with virtual vehicles, but if a collision with a particular virtual
    vehicle occurs for the first time, e is tagged with the virtual vehicle's id and flag.
}
```

As model vehicles may own any number of tags, a list (of tags) is associated with them. For any tag, if *flag* is true, it means that the model vehicle overtook or overlapped with a virtual vehicle at some point. If it is false, it means that the model vehicle came into contact with another tagged model vehicle at some point. If the model is more sophisticated and contains traffic signals for example, tags must be copied every time there is a conditional link. For example, if a tagged model vehicle reaches an intersection, and, because of its presence causes the light to switch from green to red for another lane, the tag will have to be copied to every model vehicle in the stopped lane.

5.4 Experimental Evaluation

In order to evaluate the speedup of the superimposed approach, a synthetic simulation experiment (with randomly generated input) was performed with varying numbers of virtual vehicles on a road network. These experiments do not flush redundant tags using the minimum propagation delay techniques. The implementation is sequential; a work-optimal parallel version could easily be created by partitioning the cell regions and mapping them to logical processes (LP). Each LP would execute the algorithm listed previously for the cell transitions of its vehicles. Once a vehicle exits the region of a given LP, a message is sent to the LP that owns the neighboring region of cells.

Experimental Setup

The simulated transportation network has a Manhattan style topology consisting of nine north-south and nine east-west two-way roads. The intersection-to-intersection distance measures 100 cells. Each vehicle has a randomly selected intersection as its destination and will take a shortest path to it. Model vehicles are generated at the end points of each two-way road. Two scenarios will be considered. The "low traffic" scenario will generate a model vehicle with probability 0.1 for each source cell during each timestep, while the "high traffic" scenario will use a probability of 0.5. Model vehicles have a maximum speed of 3 cells per timestep and a p=0.5 deceleration parameter. The simulation runs for 500 timesteps and uses the superimposed Nagel-Schreckenberg algorithm with the tagging mechanism. The target vehicle E_T, is released at Timestep 3 and varies its deceleration parameter (0.2, 0.25, 0.3, 0.35, 0.6, 0.65, 0.7, and 0.75), its maximum speed (2, 3, 4, and 5 cells per tick), and its destination (uniformly selected among all intersections).

Speedup vs. Number of Replications and Traffic Intensity

Figure 5.2 shows the execution times for the naive approach of running one replication per realization of E_T (**B**rute **F**orce **E**xecution **M**ode) and the superimposed approach with the tagging mechanism (**V**irtual **V**ehicle **E**xecution **M**ode). The orange and grey lines cover data points from the high and low traffic scenarios, respectively. As the number of replications increase, the VVEM execution time grows more slowly than the BFEM execution times. The

E_T travel paths were verified to yield identical results for both execution modes. At 64 replications, the speedup of VVEM (relative to BFEM) is 5.8 for the low traffic scenario, while it is 1.65 for the high traffic scenario. This is intuitive because in highly congested systems, many model vehicles will be able to overtake the virtual vehicles. As a result, more model vehicles will be tagged, and their tag propagation rate is also much higher. Subsequently, the probability of having to spawn physical clones increases. The results show that one should opt for larger batch sizes when using VVEM - especially when modeling a highly congested network. The test computer is an Intel Core i5-3550 with 16 GB DDR3 RAM running Windows 7 64bit.

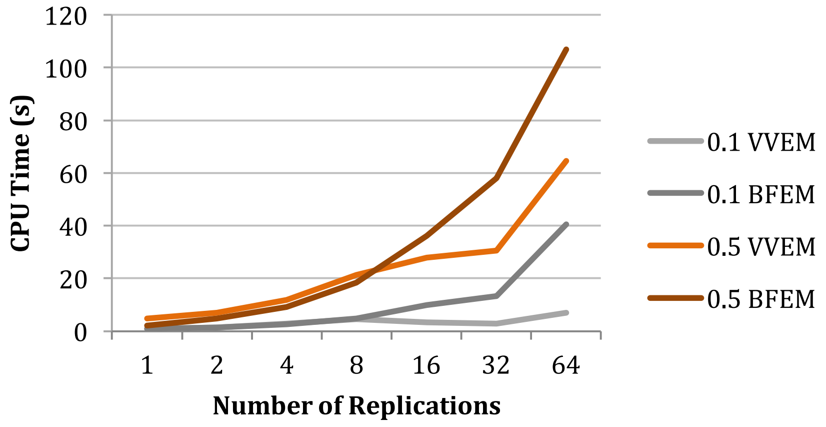


Figure 5.2: Execution time results by scenario

Speedup vs. Relative Speed Differences

Because the main VVEM run is, in most cases, only slightly more computationally intensive than a traditional replication, the VVEM speedup is equal to approximately *N/(1+C)*, where *N* is the number of replications (all N replications are executed by BFEM) and *C* is the number of physical clones triggered by VVEM. The number of triggered clones is primarily dependent on how many model vehicles are able to overtake the virtual vehicles. This metric, in turn, is directly related to the traffic intensity and the speed of the general traffic relative to the virtual vehicles. This is a result of model vehicles being tagged as they overtake the slow virtual vehicles and propagating the tag further. This increases the probability of a clone being triggered. Figure 5.3 shows the VVEM speedup for different values of the global speed limit parameter ***V*** (used by MV's) as well as the traffic scenario (high vs. low). Each scenario is run five times (in both VVEM and BFEM) for N=16 virtual vehicles for all permutations of low & high VV deceleration probability set (<0.2, 0.25, 0.3, 0.35> vs. <0.6, 0.65, 0.7, 0.75>) and a low & high VV maximum speed (<2, 3> vs. <4, 5>). The average speedup is displayed for each scenario. It is clear that a lower speed limit results in a higher speedup as the model vehicles are less likely to overtake the virtual vehicles (independent of the traffic intensity) and, thus, less likely to trigger physical clones. It is also noticeable that (comparatively) fast virtual vehicles result in a very high speedup (although it is more extreme in the low traffic scenario) as model vehicles are less likely to have to slow down and be tagged.

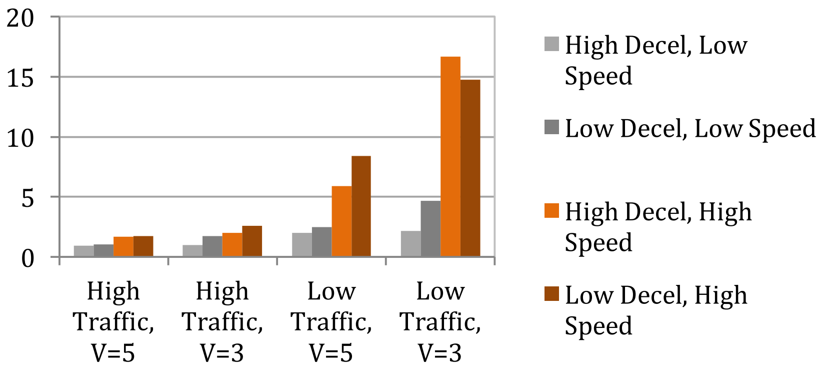


Figure 5.3: Speedup for different relative speed tendencies

Tag Memory Footprint vs. Traffic Intensity and Speed Limit

A VVEM replication uses more memory than a single BFEM replication because the tags from VV-MV collisions need to be stored as they are spread throughout the simulation environment. Figure 5.4 shows the rate of growth in memory requirements as the traffic intensity increases; no tag flushing procedure was used in those iterations. One could clear tags of VV's that have reached their destination as well as tags of MV's that cannot reach the corresponding VV based on the minimum propagation delay. The values were computed at Timestep 350 using 64 VV's.

.

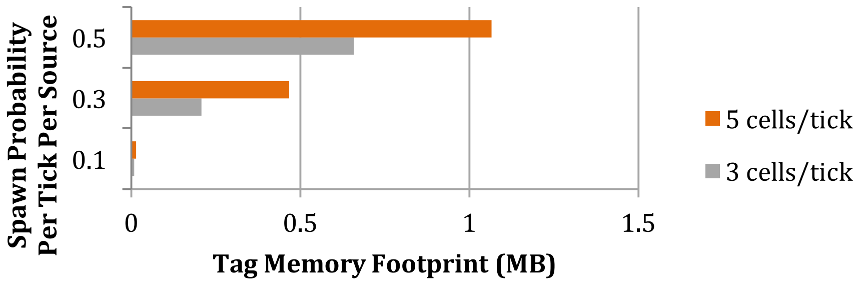


Figure 5.4: Memory requirements for storing the tags

5.5 Conclusion and Future Work

An algorithm to speed up replicated traffic simulations was presented. The concepts can also be applied to other models, but the benefits are particularly well-suited for models with similar characteristics. Three of these characteristics are the local movement of entities (triangle inequality), low connectivity (vertices with low degree), and simple output collection (travel time of E_T). The experimental results show that the execution time grows slowly as the number of E_T instances increases.

We plan to investigate what heuristics can be used to determine how and when tags should be cleared. There is an overhead from performing a check, but there are also gains from fewer comparisons and reduced storage requirements. One can also envision a hybrid between execution with and without error detection where heuristics will compute the probability of a certain error occurring (e.g., given a macroscopic analysis of the traffic intensity). It is also worthwhile to study what models benefit from the superimposed approach and its performance using real-world network configurations. Under what circumstances is it worthwhile to physically clone, rather than verify whether an error occurs via the tagging rules? There is also a bias-variance tradeoff present. The analyst may simplify the model and speed up the execution. This will cause a reduction in the half-width for fixed computation time. However, using a more sophisticated model may reduce the bias at the expense of computational time (because certain optimizations cannot be used).

6 Data Distribution Management for On-Line Simulations

With the growing use of mobile devices, power aware algorithms have become essential. Data distribution management (DDM) is an approach to disseminate information that was proposed in the High Level Architecture (HLA) for modeling and simulation. This chapter explores the power consumption of mobile devices used by pedestrians in an urban environment communicating through HLA DDM services operating over a mobile ad-hoc network (MANET). The computation and communication power requirements of Grid-Based and Region-Based implementation approaches to DDM are contrasted and quantitatively evaluated through experimentation and simulation.

6.1 Introduction

Dynamic Data Driven Application Systems (DDDAS) are applications that continuously monitor, analyze, and adapt operational systems in order to better assess and/or optimize their behavior. Applications arise in many areas such as transportation, medicine, disaster management, and manufacturing, among others (Darema 2004). Many DDDAS applications involve sensing and computation on mobile devices, utilizing communications through wireless networks. Power consumption in these applications is a major concern because battery life often limits the effectiveness of DDDAS applications utilizing mobile platforms.

Mobile computing is a research area that is receiving much interest with the rapidly growing use of mobile devices. A majority of the U.S. population uses smartphones. Popular operating systems such as android, iOS etc. have emerged and are constantly undergoing changes. Most of these operating systems support sensors such as GPS, photo and video camera, accelerometer and also provide location and map based services, providing a rich source of data for on-line applications.

Mobile ad-hoc networks (MANETS) provide communication services for mobile devices without relying on fixed infrastructure. They provide communication services among a set of mobile and fixed location devices and have received much attention by the research community.

Crowdsourcing has begun to emerge as a paradigm for collecting information. Crowdsourcing is the practice of obtaining needed services, ideas, or content by soliciting contributions from a large group of people and especially from the online community rather than from traditional employees or suppliers. A well-known example is the 2009 DARPA Balloon Challenge where teams were tasked with finding 10 red weather balloons placed across the U.S. using social media to gather information (Tang et al. 2011). The winning team from the Massachusetts Institute of Technology correctly located the ten balloons in less than nine hours. Their winning strategy included a financial incentive for recruiting members and used a recursive pyramid money scheme to keep members interested and to continue to recruit other members until the balloons were found. The DARPA challenge demonstrated that certain surveillance tasks could be accomplished much more rapidly and at a much lower cost than traditional means for collecting information.

Mobile on-line simulations have also received much attention in recent years. For example, in mobile traffic simulations fed by online data sources were used to predict congestion in urban transportation systems (Fujimoto et al. 2007). Such simulations find use in many DDDAS applications, as discussed earlier.

Here, we are concerned with DDDAS applications that use techniques such as crowd sourcing, MANETS and predictive simulations operating on mobile platforms to analyze operational systems. Specifically, we are concerned with the amount of power consumed in such applications, e.g., as a pedestrian carries a mobile device throughout the system. We assume that MANETS are used to disseminate information throughout the system.

Data distribution management (DDM) is a set of services defined in the High Level Architecture (Rak and Van Hook 1996) to distribute information in distributed simulation environments. DDM services are implemented by Run-Time Infrastructure (RTI) software. Several different approaches to implementing the DDM services have been proposed including grid-based implementations, region-based implementations, and hybrid approaches that utilize a combination of ideas from the grid and region-based approaches (Tan et al. 2000). These approaches have certain computation and communication requirements, as will be discussed later. However, to our knowledge, DDM services have yet to be examined from the standpoint of power consumption requirements. This is the focus of the work described here. We examined the grid- and region-based approaches for this study because they represent two very different approaches to DDM that have been widely reported in the literature. Specifically, we focus on power consumption issues in using DDM services for DDDAS applications including crowd-sourced information that is used to drive mobile distributed simulations.

The rest of this chapter is organized as follows. Section 6.2 reviews different approaches to implementing the HLA Data Distribution Management services, focusing on the Grid and Region-based methods that are studied here. Section 6.3 describes tradeoffs that arise between grid-based and region-based implementations with respect to power consumption for computation and communications. Factors of the implementation that impact power consumption include the grid cell and region size. Section 6.4 describes the application scenarios used in our experiments to compare power consumption for these two approaches. Section 6.5 presents the results of power consumption measurement of DDM computations on a representative mobile device. Section 6.6 presents the results of power consumption that takes place during communication when grid-based DDM method is implemented over a MANET using the NS3 simulator. Section 6.7 summarizes the results and presents conclusions and areas of future work.

6.2 Approaches to Data Distribution Management

Data Distribution Management services are defined in the HLA to reduce traffic flow over the network. The DDM services define an N-dimensional coordinate system known as the routing space. Publication and subscription regions are defined within this routing space to characterize the information contained in messages and to specify what information federates are interested in receiving, respectively. Typically, the routing space represents a geographical area. Data producers specify which regions are relevant to a message by associating a publication region with each message. Data receivers specify the areas of the routing space in which they have interest by associating subscription regions with their data subscriptions. If the publication region associated with a message overlaps a federate's subscription region, the message is routed to that federate. The grid-based and region-based implementations are well known approaches to realizing the HLA DDM services.

Grid-Based Approach

The grid-based DDM approach divides the routing space into fixed sized grid cells. A s group is assigned to each grid cell. When a federate subscribes to a subscription region it joins each multicast group whose cell overlaps with the subscription region. When a message is sent, it is transmitted to each multicast group associated with a grid cell overlapping with the publication region associated with the message. A matching computation must take place to determine which grid cells overlap with each publication region to determine the multicast groups that are used to transmit the message.

Figure 6.1 shows an example of the grid-based DDM approach. In this example the publisher region P1 would publish messages to all federates whose subscription region include the grid cells that overlap with P1, i.e., cells 10, 11, 15, 16, 20, and 21. Similarly, S1 overlaps with grid cells 1, 2, 6, 7, 11, 12, 16 and 17. Note that in this case, federates subscribed to S1 will receive two identical copies of the message – messages sent to groups 11 and 16. Federates subscribed to region S2 will not receive the message because they are subscribed to groups 13, 14, 18, 19, 23 and 24. It may be noted that federates may receive extra (irrelevant) messages, i.e., it is possible a message will be received even though the publication and subscription regions do not overlap. This occurs if the non-overlapping regions have a grid cell in common. If communications are reliable, the grid-based approach will result in all federates whose subscription region overlap with the publication region of the message receiving at least one copy, however as noted earlier, duplicate or additional (irrelevant) messages may be received, and must be filtered at the receiver.

The number of duplicate and irrelevant messages that are sent and received by federates is of particular concern here because these result in unnecessary power consumption. The size of the grid cells can have a large impact on the cost of sending data and receiving irrelevant data. D.J. Van Hook and S.J. Rak (1996) explored the effect of grid cell space sizes and the effect it has on multicast group assignment and communication cost. They concluded that communication cost becomes increasingly greater the larger the size of the grid cell because it causes an increase in irrelevant and duplicated messages. This issue has been explored by Tan and Xu who ran several experiments on the sending of data using grid-based DDM. They proposed an alternate approach called Agent-based DDM to decrease the number of irrelevant messages sent across a network (Tan et al. 2001).

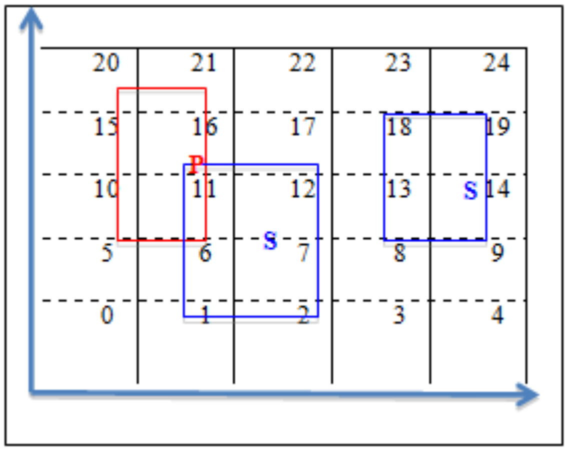


Figure 6.1: Grid-Based DDM

To implement this approach to DDM the space was divided into fixed size grid cells. Federates were created and added to the space. Each federate was assigned a single publication and a single subscription region. The grid-based matching computation executes the following:

```
For all grid cells:
  For all federates:
        Check if Publication/subscription regions overlap with the grid cell
        If regions overlap, add the respective grid_id to the pub/sub multicast group of the respective grid cell.
```

On a publication (or subscription) region update, the program first compares the coordinates of the updated publication (or subscription) region to the coordinates in its publication (or subscription) multicast group. All the grid cells from the multicast group for whom coordinates do not overlap are removed from the group. The next step is to compare the coordinates of the updated region with remaining grid cells and add all those grid cells to the multicast group for whom the coordinates overlap.

Region-Based Approach

The region-based approach does not use grid cells. Rather, a multicast group is defined for each publication region. Federates join the multicast groups associated with publication regions that overlap the subscription regions to which the federate is subscribed. This is a direct implementation of the DDM services, and is sometimes referred to as the "brute force" approach to DDM.

Region-based matching is clearly more efficient than the grid-based approach in terms of message communications in that there are no duplicate or irrelevant messages. Publication regions are checked against subscription regions to identify overlaps and information is then distributed when overlaps exist. Region-based DDM has been implemented on the MAK high performance RTI by Wood (2002). Implementation of region-based DDM does incur additional messages depending on the implementation. For example, if a centralized implementation is used, region changes must be communicated to the server in order to determine the new composition of the multicast groups.

The drawback of the region-based approach is a matching computation must be performed to determine which subscription regions overlap with which publication regions. This computation must be repeated each time a publication or subscription region changes.

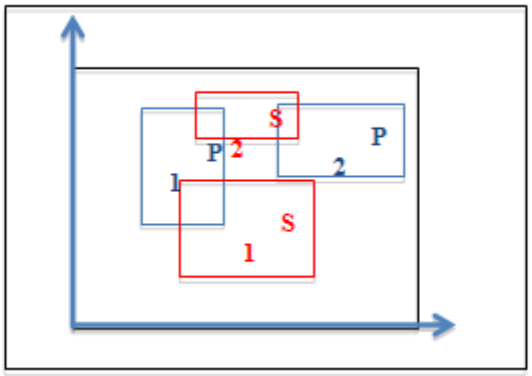


Figure 6.2: illustrates an example of Region-Based DDM. In this example P1 publishes information to federates who are subscribed to regions S1 and S2. P2 publishes information to federates who are subscribed to region S2.

In this implementation federates were added to the entire space. Each federate was assigned a publication as well as a subscription region. The matching algorithm executes the following:

```
For all federates (F1):
  For all federates (F2):
      Check if Publication region of F2 overlaps with the subscription region of F1.
      If regions overlap, add the respective federate_id to the publication multicast group of F2.
```

On a publication region update, the matching algorithm is executed only for the federate with the updated region.

On a subscription region update, the updated subscription region is compared against all publication regions. If the regions overlap, the program adds the federate to the multicast group associated with the federate's publication region. If the regions do not overlap, the federate (if present) leaves the corresponding multicast group.

6.3 Power Consumption Tradeoffs

Power consumption of the DDM services using the grid-based and region-based approaches is somewhat complex because it must consider several tradeoffs. The region-based approach must consume power to compute overlaps among publication and subscription regions. This computation is $O(N^2)$ where N is the total number of publication/subscription regions used in the entire federation. Whenever a publication (subscription) region changes the new region must be compared against all other subscription (publication) regions to determine overlaps with the new region. This computation could be reduced by overlaying a grid structure onto the routing space, however, this hybrid approach to the region-based scheme is not investigated here. In addition, if the DDM mechanism is implemented centrally at one device, communications with this device is necessary whenever regions change, necessitating additional power consumption. If the DDM implementation is distributed, e.g., across a set of devices, communications among these devices is similarly required whenever regions change to coordinate management of the groups.

The grid-based approach requires computation to determine overlaps between regions and grid cells when a region changes, however, this computation is O(1). This will be significantly less than that for the region-based approach. Further, the grid-based approach allows a fully distributed implementation without the need for coordination messages that are necessary in the region-based approach when regions change. However, the grid-based approach incurs additional power consumption to transmit irrelevant and duplicate messages, something that is not required in the region-based approach. The amount of additional communication will depend on factors such as the grid cell size, the number of federates, the size and location of publication and subscription regions, and the number of hops between communicating nodes in the MANET.

The tradeoffs among these factors clearly depend on the specific scenario that is used and particulars of the DDM design. In the following we empirically evaluate these questions through a combination of measurement and simulation. The scenario used in this work is based on a college campus where individuals walk from one location to another. A two-dimensional routing space is used that corresponds to the college campus itself. Mobile units publishing data are assumed to send information concerning observations made by the mobile unit, i.e., their publication region corresponds to an area not far from their current location. Subscription regions represent areas of interest to a particular mobile device and are typically further removed from the location of the subscriber federate.

6.4 Scenario and Experimental Setup

Our scenario consists of federates (mobile devices) moving throughout the simulated college campus. The federates report observations concerning an object, e.g., a vehicle that the federates might be attempting to monitor. Federates report information or receive information, e.g., observations made by a particular mobile device as it moves throughout the environment. This simulated environment is approximately 1280 meters by 1280 meters. The simulation consists of two major components, the Data Distribution Management component that manages federate publication and subscription regions and determines the composition of multicast groups and the NS3 (network simulator) component that routes packets among federates over a MANET.

The Data Distribution Management component was created for three different sets of experiments. The experiments determine power consumption for three different grid cell sizes (10X10, 20X20, and 40X40). The DDM grid-based method consists of creating grid cells that are overlaid on top of the simulated area. This grid cells are then used to assign federates to multicast groups based on the publication and subscription regions. Each federate has a single publication and subscription region. Each federate's publication region covers an area that is a half of a mile away from their current location and their subscription region covers an area that is a mile away from its current location. Each grid cell has its own multicast group. A federate sends messages to, or joins a particular multicast group based on the location of their regions. The federate joins every multicast group of the respective grid cells that their regions cover. These assigned multicast groups are used to determine who receives packets when a publication is made. Changing federate region locations take place during the simulation when an update event occurs. The DDM grid-based method then determines the federate's new assigned multicast groups based on their new region locations. Federates first leave their current multicast group before joining a new multicast group. These new region locations are determined based on the current location of the federate at the time during the simulation when this event takes place, the regions are reassigned using the coordinates of the current location of the federate. The grid-based DDM mechanism handles assigning all of the multicast groups that NS3 uses in order to multicast packets to the federates.

The amount of power consumed for computation is determined through direct measurement of the associated algorithms on a mobile device. The amount of power consumed for communications over the MANET is determined through simulation using the NS3 network simulator.

NS3 is used to send packets to respective multicast groups. The NS3 component of the experiment consists of creating a wireless ad hoc network that uses the OLSR (Optimized Link State Routing) routing protocol to route packets to federates. Packets sizes of 1000 bytes were used for these experiments.

The federates follow a mobility pattern that reflects the movement of pedestrians moving through the campus. Mobility files were created using the mobility scenario generation tool Bonnmotion (Aschenbruck et al. 2010). The SMOOTH model was used in order to reflect mobility patterns of pedestrians moving through the campus. SMOOTH is a simple way to model human movement. SMOOTH consider walking behavior to be self-similar. This principle is the grounding foundation for generating mobility models, since patterns of human walking behavior are considered self-similar the model can be used for any scale of larger models. SMOOTH is a simplified version of the original mobility model SLAW, which produces the same mobility patterns with the need of more parameters (Lee et al. 2009). The model generates mobility patterns based on seven features: the distribution of the movement of nodes, the distribution of nodes, the pause time distribution, popular waypoints in the network, closest waypoints that mobile nodes tend to visit first, the distribution of mobile nodes in a non-uniform network, and mobile nodes with common interest form communities of interest (Munjal et al. 2011). The parameters of SMOOTH that reflect the dynamics of the university campus used in this study were used as inputs for our mobility trace files that include: the size of the area and clusters that represent common places on campus were people tend to linger for long periods of times. The clusters in the experiments represent a Commons area, the student center, recreational center, a commercial region, and an instructional center on the Georgia Tech campus.

The distance of the waypoints between the clusters were a part of the inputs used in order to generate accurate trace files that reflect the system under investigation. Once these trace files are created they are then stored in trace files and used as input to NS3. The files are then imported into NS3 to be used as the mobility model for the simulation. Once the mobility model was set for the simulation the closest federate to the vehicle being observed sends a packet. The nodes that receive the message are those that have been determined by the DDM component of our experiment that their subscription region overlaps with grid cells that are a part of the federate's publication region.

The NS3 Wi-Fi radio energy model was used to measure the amount of power consumed by devices attached to each federate. Every ten seconds during the simulation an update event occurs. During this update federates update their current location and create new publication and subscription regions. The federate that is now closest to the object being monitored then becomes the new source for publishing a message to those federates that are members of the multicast groups that contain the source federate publication regions and overlapping subscriber regions. The amount of energy consumed by these devices while federates are sending and receiving packets during the simulation are continuously added until communication between federates end. These measurements are then converted to watts to reflect the amount of power consumed during communication.

6.5 Power Consumption: Computation

The associated DDM computations were implemented on a Google Nexus 4 smart phone, using PowerTutor to measure power consumption for the android device (Gordon et al. 2013). Power Tutor is a smartphone application that measures the power consumption of CPU, Wi-Fi, 3G, and applications running on Google based android devices. The focus of this experiment only considers power consumed by the CPU.

The routing space was assumed to be a 400 x 400 unit space. For the grid-based implementation, the routing space was divided into 400 grid cells of 20x20 units each. The number of federates were varied from 100 to 1000 incrementing by 100 for each experiment. The publication and subscription regions of every federate were chosen randomly within the routing space. The code for both implementations was written in Java using the Android Development Tool (ADT) plugin for the Eclipse IDE and was tested on the Nexus 4 smartphone while monitoring the CPU power consumption using PowerTutor application. The consumption results obtained from these measurements are shown in Figures 6.3 (energy) and 6.4 (power).

From Figure 6.3 it is clear that the region-based implementation consumes much more energy than the grid-based approach with the same parameters. This result is expected since the grid-based method reduces the computation cost involved by dividing the routing space into grid cells. We also observe that in the grid-based method, the incremental power consumption is proportional with the number of federates and this can be attributed to the fact that the number of computations involved is directly proportional to the number of federates.

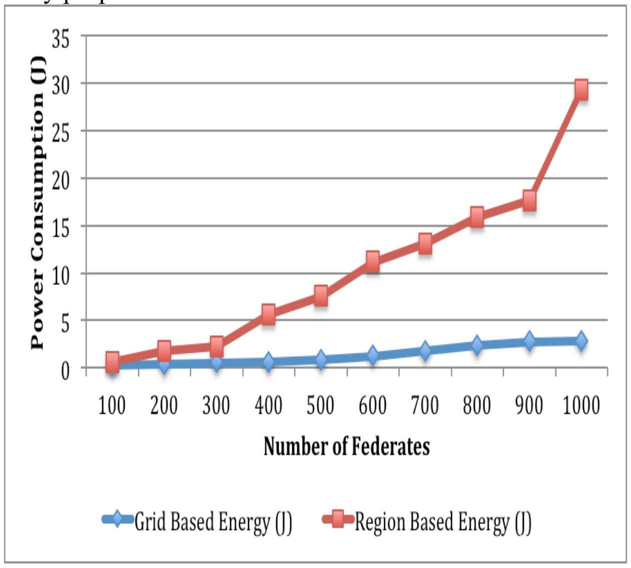


Figure 6.3: Energy Consumption results for computation in region-based and grid-based DDM.

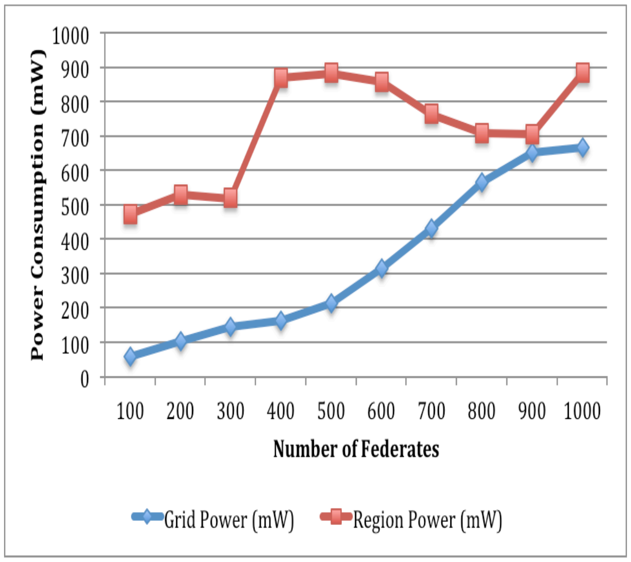


Figure 6.4: Power consumption results for computation in region-based and grid-based DDM.

On the other hand, in the region-based method, though the power consumed increases initially with the number of federates, it does not increase significantly beyond a few hundred federates. It should be kept in mind that the power is calculated in mW and thus even though the graph indicates that the power consumed in the region-based method does not significantly change, the total amount of energy consumed does increase as is shown in the Figure 6.3.

The computational cost in the region-based method is O(N^2), where N is the number of federates. The above result is consistent with this observation.

6.6 Power Consumption: Communications

Grid Cell Size Experiment

The previous section compared the power consumption for computation using the grid-based and region-based implementations of DDM. Here, we examine the power consumed for communications, specifically for the grid-based approach as the size (and thus the number) of grid cells is varied. We also examine the power consumed for communications for the grid-based approach as the grid cell size is varied for different publication and subscription region size..

The number of multicast groups increases with the number of grid cells. This will affect the number of irrelevant messages (i.e., messages sent to a federate whose subscription region does not overlap with the publication region) and duplicate messages. Increasing the number of grid cells results in a finer grid cell structure that may reduce the number of non-overlapping publication and subscription regions that have a common grid cell, resulting in fewer irrelevant messages. On the other hand, if the publication and subscription regions are sufficiently close to cause irrelevant messages, having a larger number of grid cells (multicast groups) may result in magnifying the number of irrelevant messages. This is apparent when one considers the case of subscription and publication regions that do not overlap, but have a long edge in parallel with each other that are in close proximity.

Concerning duplicate messages, consider a publication and subscription region that do overlap. Smaller grid cells will result in more multicast groups in common with the overlapping portion of the subscription and publication regions, which would, in turn, increase the number of duplicate messages.

Another factor that impacts power consumption concerns the fact that the DDM mechanism is operating over a MANET. The number of hops for a single message will depend on the location of the communicating federates and mobile nodes forwarding the message. Some messages will consume more power than others because they must traverse several hops to reach their final destination.

We simulated the grid-based Data Distribution Management approach using NS3 with from 10 to 90 federates for different grid cell sizes in order to investigate these tradeoffs and estimate the amount of power that was consumed (NS-3 2011). The grid cell sizes used were 10X10, 20X20, and 40X40. Our results, shown in Figure 6.5 indicate that increasing the number of grid cells causes an increase in power consumption. We believe this is due to an increase in duplicate messages, as described earlier. One would anticipate that as the number of federates increases the amount of power consumption would also increase, Figure 6.6 indicates that this case is true. The experiment that used the 10X10 grid cell size produces the least amount of power consumption.

Figure 6.7 shows the number of packets sent and received for differing number of federates at all three different grid cell sizes. It is seen that power consumption tracks the number of messages sent and received, confirming one's intuition that power consumption for these experiments largely depends on the amount of communication. For the 10X10 case fewer packets are sent and received resulting in reducing the amount of power consumption in comparison to the 20X20 and 40X40 experiment. This result is a reflection of the small amount of communication taking place. With the cost of less power consumption comes the loss of efficiency. It may be noted that this data does not take into account the power required to filter duplicate and irrelevant messages, which should also increase in proportion with the amount of communication.

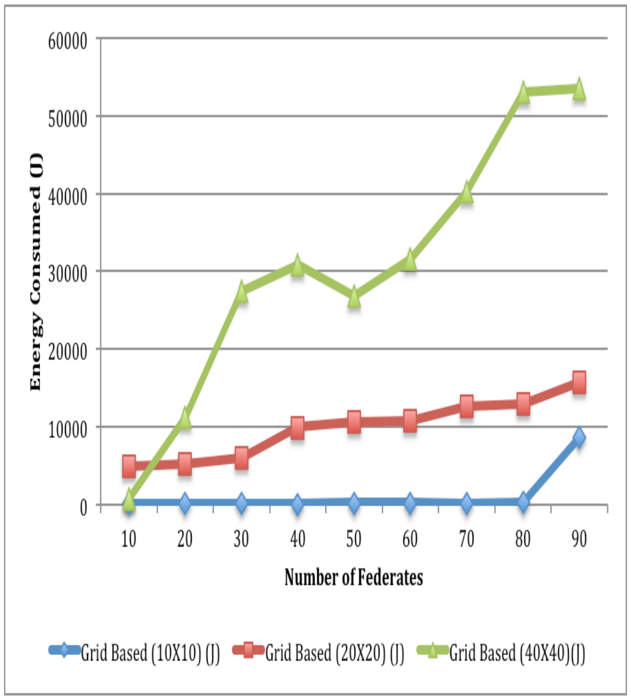


Figure 6.5: DDM Energy Consumption for communications

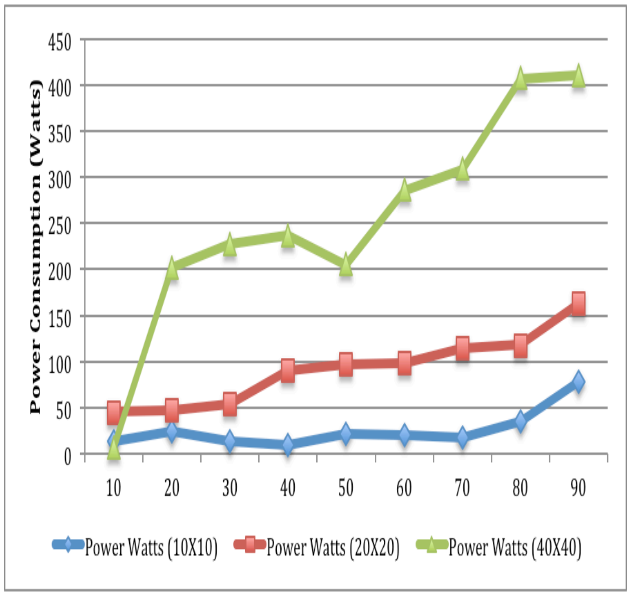


Figure 6.6: DDM-Grid Power Consumption for communications.

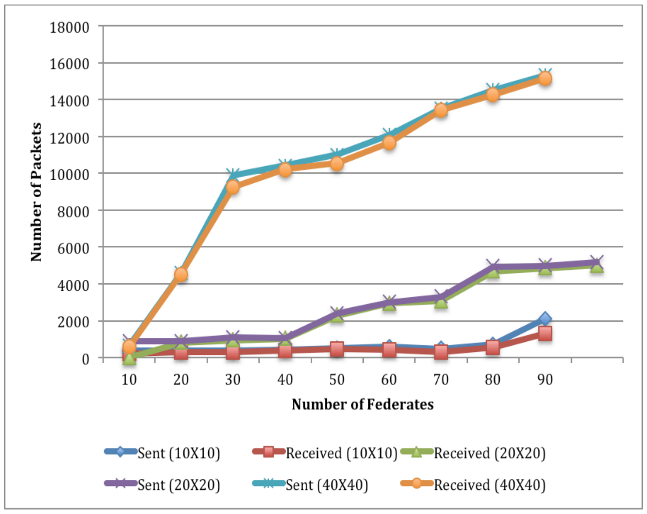


Figure 6.7: DDM Grid Packets Sent and Received.

The results from our experiments reflect that power consumption during communication for grid-based Data Distribution Management is greatly impacted by grid cell size. A larger number of grid cells increases the number of multicast groups which can increase the amount of communication and result in increased power consumption.

Publication and Subscription Region Size Experiment

This experiment was conducted using the same set up as the initial experiment described in Section 6.1 with a modification to the region sizes for each federate, the subscription region size was decreased to half of a mile and the publication region was decreased to a quarter of a mile. The purpose of this experiment is to see the impact that region sizes have on power consumption. One would anticipate that the size of a federates publication and subscription region has a significant affect on the amount of power consumed. Our results are consistent with this expectation as shown in Figure 6.8.

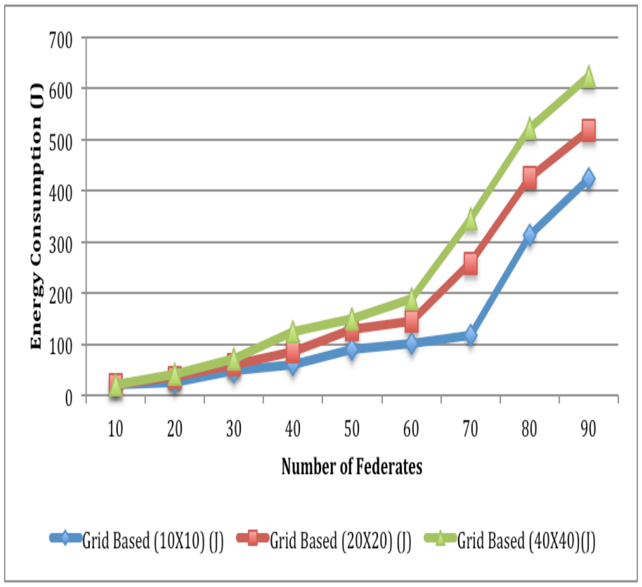


Figure 6.8: DDM-Grid Energy Consumption for communication from smaller region scenario

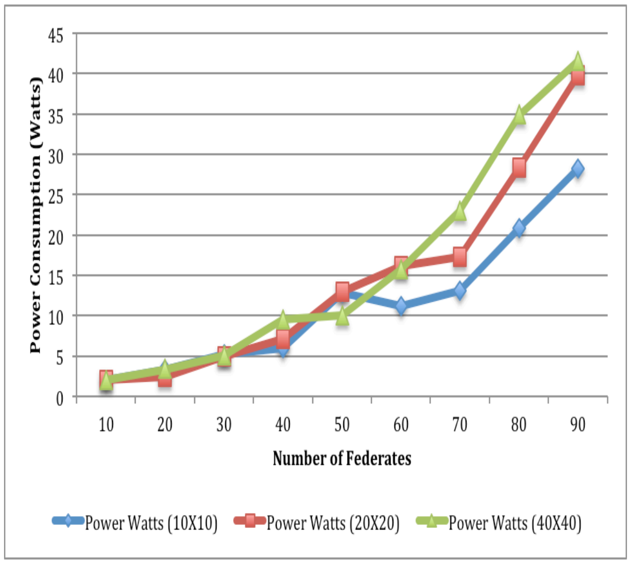


Figure 6.9: Power Consumption for DDM-Grid Based Communication for smaller region scenario

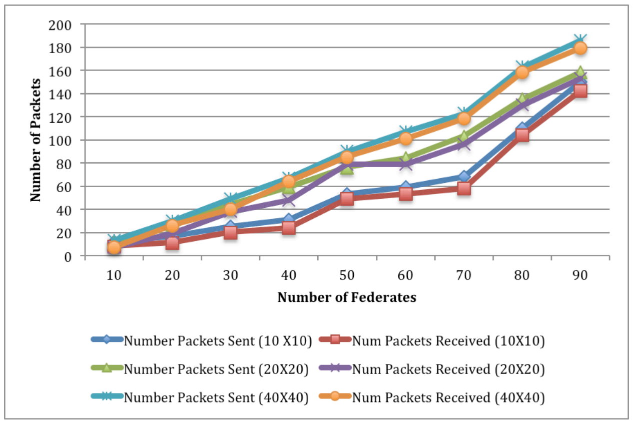


Figure 6.10: DDM Grid Packets Sent and Received with Smaller Region Sizes

Figure 6.8 shows the amount of energy that was consumed for different grid sizes using smaller regions. As before, Figure 6.8 shows that the amount of energy consumed increases as the grid cell sizes decreases. However, the difference in energy consumption among the three different grid cell sizes is much less than before. This result can be attributed to the fact that the number of multicast groups to which each federate is now subscribed and publishes has significantly decreased with a smaller region size. A federate now sends and receives fewer packets due to the smaller region sizes. Figure 6.9 shows the amount of power consumed presents a strong positive correlation between the amount of power and the change in grid cell size. Figure 6.10 shows that the number of packets sent and received decreases significantly compared to Figure 6.7 where publication and subscription region were twice as large. Comparing these two experiments we can draw the conclusion that the larger the publication and subscription region size the more power will be consumed during communication.

6.7 Conclusions and Future Research

Communication and computation cost are the key factors to choosing any algorithm when power consumption is an important issue. Our experiments evaluate the tradeoffs of choosing grid-based or region-based DDM when determining the effects that their methods cause on power consumption. Region-based DDM has a $O(N^2)$ computation cost. The results of our experiments reflect that when comparing overlaps of publication and subscription regions when using region-based DDM the power consumption increases at a significant rate because of the direct comparison of each region against all other regions. Grid-based DDM requires much less computation. Because all grid cells have their own designated multicast group minimal computation is needed to determine which federates will receive information when a publication is made. Communication has an inverse affect on grid-based DDM power consumption. As the size of the grid cell decreases the amount of power consumed increases. Although little computation is need for grid-based DDM much more communication takes place when a publication is made because of the large number of multicast groups. Choosing grid cell sizes has a direct impact on power consumption because a larger number of grid cells may result in more power consumed. Although more power is consumed with a larger number of cells the efficiency of communication may be reduced. Finding the optimal grid cell size in terms of power consumption is an area of future research. As the region size decreases there is less communication and correspondingly less power consumption. This highlights the need to restrict the size of publication and subscription regions to the minimum required size in order to keep power consumption low.

When choosing which Data Distribution Management method is more power efficient with respect to power consumption both computation and communication costs should be considered. Region-based DDM methods have many tradeoffs with respect to power consumption for computation and communications. Our experiments reveal the effects that DDM computation has on power consumption when comparing grid-based and region-based methods. Region-based DDM consumes significantly more power than grid-based DDM for computation as the number of federates is increased. Our experiments indicate that when evaluating the effects that grid cell size plays on power consumption, the communication cost is affected by several factors. The largest factor is the number of packets that were sent and received. The decrease in grid cell size results in an increase in the number of multicast groups causing an increase in the number of packets that must be sent and received during communication between federates. Our experiments also indicate that the decrease in federate publication and subscription region size results in fewer message communications and less power consumption. As before, the number of multicast groups affects the number of packets that must be sent and received during the execution, however, the impact on power consumption is less compared to the case using larger region sizes.

This work represents a preliminary investigation of the power consumed by DDM implementations. A more comprehensive study of the power consumed by grid-based and region-based DDM approaches is needed. A theoretical model that can predict the amount of power consumed by these approaches may be useful. Other future work includes evaluating the power consumed by other Data Distribution Management methods for computation and communication. For example, the hybrid method that combines grid cells with the region-based approach may reduce the power consumed for computation (Tan et al. 2000). Evaluation of power consumption needs for specific application scenarios is another area of future research. As mobile computing, MANETs, crowdsourcing, and DDDAS approaches become more common, power consumption issues will become an increasingly important concern.

7 Energy Consumption in Distributed Simulations

Power and energy consumption are important concerns in the design of high performance and mobile computing systems, but have not been widely considered in the design of parallel and distributed simulations. The importance of these factors is discussed and metrics for power and energy overhead in parallel and distributed simulations are proposed. Factors affecting the energy consumed by synchronization algorithms and software architectures are examined. An experimental study is presented examining energy consumption of the well-known Chandy/Misra/Bryant algorithm executing on a peer-to-peer mobile computing platform and compared with a centralized client-server approach using the YAWNS synchronization algorithm. Initial results concerning queueing network simulations are also presented. The results of this study suggest that existing distributed simulation algorithms require a significant amount of additional energy compared to a sequential execution. Further, different synchronization algorithms can yield different energy consumption behaviors.

7.1 Introduction

Power consumption has become a major concern for many parallel and mobile computing applications. The need to reduce energy use is clear in mobile and embedded computing where reductions result in increased battery life or enable the use of smaller batteries thereby reducing the size and weight of devices. In high-end computing energy consumption is a dominant cost associated with operating large data centers and supercomputers, and a substantial amount of effort has gone into developing techniques to mitigate this expense. Power consumption has become the key factor preventing substantial further improvements in clock speed and now limits computer performance. It has been cited as a major obstacle to creating supercomputers yielding exascale performance. Despite the importance of power and energy in computation today, very little attention to date has focused on understanding and developing techniques to minimize power and energy consumption in parallel and distributed simulations.

Energy is the capacity of a system to perform work. It is typically measured in units called joules where one joule is the work performed by an electrical circuit to move a charge of one coulomb through an electrical potential difference of one volt. Power is the amount of energy consumed per unit time with one watt of power defined as the expenditure of one joule of energy per second.

Minimizing energy usage and power consumption are not the same thing (Unsal 2008). For example, decreasing the clock rate of the processor can lead to less power consumption. However, this will usually lead to longer execution times and an increase in the total amount of energy needed to complete a given computation. Battery operated devices are *energy-constrained* systems because they operate with a finite amount of available energy; thus a design goal might be to minimize the amount of energy utilized by the computation as a whole, subject to certain execution time constraints. On the other hand, in *power-constrained* systems such as supercomputers and data centers the amount of available energy is effectively unlimited, but a design goal may be to minimize the amount of time required to complete the computation given a certain maximum level of power consumption, or to minimize power consumption, subject to certain execution time constraints.

Power-aware and *energy-aware* systems are those where power or energy consumption is a principal design consideration. For example, power-aware systems may utilize techniques to change the system's behavior based on the amount of power being consumed. Energy-aware systems may modify the operation of the system based the amount of energy remaining in batteries.

It should be noted that minimizing execution time does not necessarily result in minimal energy consumption. Energy consumption is affected by many factors, e.g., the operation of the memory system, the number and complexity of computations performed by arithmetic circuits, and importantly, the amount of inter-processor communication that is required. A parallel or distributed computation that executes in a shorter amount of time may consume both more energy and more power if more communications are required.

Power- and energy-aware computing is increasing in importance for parallel and distributed simulation systems and applications. The main contribution of this research is to highlight the increasing importance of power and energy consumption in parallel and distributed simulations. We propose performance metrics to quantitatively assess energy consumption in parallel and distributed simulations and report initial empirical measurements of the energy consumed by concervative synchronization algorithms.

The next section describes a motivating application to highlight the relevance of energy consumption in distributed simulation. This is followed by a discussion of related work in this area by briefly surveying power- and energy-aware computing. Metrics for measuring energy and power consumption in distributed simulations are then proposed. The two distributed simulation systems examined in this study – a peer-to-peer system using the Chandy/Misra/Bryant algorithm and a client-server architecture using the YAWNS algorithm are then described. The experimental configuration and benchmark programs used in this study are presented, followed by a discussion of the power measurement methodology that is used. Experimental results are then presented and discussed, followed by concluding remarks and discussion of areas requiring further investigation.

7.2 A Motivating Application

Embedded mobile distributed simulations can be used to create adaptive sensor networks to monitor dynamically changing physical systems. For example, consider a collection of small, battery-operated unmanned aerial vehicles (UAVs) tasked with monitoring a physical system, e.g., tracking a set of vehicles moving throughout a city, monitoring the spread of a forest fire, or assessing the dispersion of a hazardous chemical plume following an accident (see Figure 1.1). Assume each UAV is equipped with sensors, an on-board computer, and wireless communications. Each UAV may initially be assigned to monitor a certain geographical area. It collects information concerning the state of the physical system in its immediate vicinity. Collectively the team of UAVs may then execute a distributed simulation to project the future state of the system, e.g., to project the location of the fire some time into the future in order to determine how best to relocate the UAVs in order to continue monitoring its spread. In some cases more UAVs may be assigned to monitor regions of particular interest, e.g., areas with higher traffic congestion, a larger density of fires, or higher concentrations of chemicals, leaving fewer UAVs to monitor areas projected to be less important to the monitoring activity. One can envision other similar surveillance applications involving teams of people carrying handheld devices or autonomous battery-powered ground vehicles.

Adaptive sensor networks such as these could utilize centralized computing capabilities where sensors report information back to a command center where predictive simulations could be executed and the network reconfigured accordingly. Placing the simulations within the sensor network itself offers several advantages. First, it reduces or eliminates reliance on connectivity to the central command center, mitigating a potential point of failure. Further, a distributed implementation enables greater scalability than the centralized approach; one can envision many teams of UAVs that collaborate to monitor larger scale systems than would otherwise be possible. In certain applications embedding the simulation within the physical system itself enables faster response time for latency-critical applications.

Systems such as these are referred to as dynamic data-driven application systems (DDDAS) (Darema 2004). DDDAS involves incorporating live data from instrumented systems into executing applications in order to optimize the system and/or steer the measurement process. The DDDAS paradigm involves repeatedly executing a processing cycle of (1) sense, (2) predict future system states, and (3) adapt or reconfigure the system to optimize the physical system or to continue the monitoring process. DDDAS has been studied in a variety of applications. Our focus here is concerned with using embedded distributed simulation to implement the second step of the DDDAS processing cycle. It is clear that for these situations, energy consumption is an important concern when the simulation operates within battery-operated mobile devices. Exploitation of the DDDAS paradigm in UAVs is an active field of study and are discussed in (for example) (Kamrani and Ayani 2007; Madey et al. 2012).

7.3 Related Work

There is a substantial literature in power- and energy-aware computing systems, and a variety of techniques that may be employed. Dynamic voltage and frequency scaling (DVFS) is concerned with altering the voltage and/or clock speed of the processor by taking into consideration energy and performance constraints (Ge et al. 2005; Freeh et al. 2007; Hua nd Qu 2003). Dynamic power consumption in CMOS circuits is proportional to FV^2 where F is the frequency at which the circuit is clocked, and V is the power supply voltage. Several commercial microprocessors support modification of the processor's frequency and voltage to trade off power consumption and performance. Scheduling algorithms for embedded systems have been designed to balance energy saving with meeting real-time deadlines, e.g., see (Saewong and Rajkumar 2003;Quan and Hu 2001; Cho et al. 2011). Processors also commonly provide different modes of operation that utilize different amounts of power. Some work examines the utilization of these modes of operation, sometimes in conjunction with DVFS (Hoeller et al. 2006; Bhatti et al. 2010; Niu and Quan 2004). Other research examines issues such as predictive modeling of energy and power (Czechowski and Vuduc 2013;Williams et al. 2009; Keckler et al. 2011) and embedded systems evaluations (Grasso, et al. 2014; Rajovic et al. 2013; Rajovic et al. 2013; Stanisic et al. 2013).

To date, work in power and energy aware computing has largely focused on low-level aspects of the computing system. Existing work focuses on effectively utilizing specific hardware capabilities, the development of operating systems and compilers, and communication protocols (e.g., routing algorithms in sensor and ad hoc networks) to reduce energy usage.

Only a modest amount of work to date has addressed energy and power consumption in parallel and distributed simulations. Some work has focused on characterizing power consumption for scientific computing applications (Dongarra et al. 2012; Feng et al. 2005; Ge et al. 2010; Esmaeilzadeh et al. 2012). One early effort examined power consumption for disseminating state information in distributed virtual environments, highlighting dead-reckoning algorithms and tradeoffs between state consistency and power consumption (Shi et al. 2003). More recent work examined power consumption related to the implementation of data distributed management (DDM) services defined in the High Level Architecture (Neal et al. 2014). To our knowledge, no prior work has examined power and energy consumption of synchronization algorithms for parallel and distributed simulations, the primary focus of the work described here.

7.4 Energy and Power Metrics

In general, the three key metrics of greatest interest are the amount of energy, power, and time required to complete a computation. These metrics may be traded off against each other. For example, as discussed earlier one can trivially reduce power consumption by reducing clock frequency at the expense of increased execution time. In high performance computing contexts both execution time and power consumption are of interest, and balanced based on constraints such as a maximum level (ceiling) of power consumption. Similarly, in real-time applications both energy consumption and maximum execution time are important for best use of system resources while still meeting real-time constraints. For this reason the product of energy (or power) and execution time, referred to as the energy-delay product, is sometimes used as a metric that simultaneously considers both energy consumption and execution time (Gonzalez and Horowitz 1996).

A central concern here is the amount of additional energy consumed by the parallel/distributed execution that takes into account parallel/distributed computing overheads such as interprocessor message communication and synchronization. For this purpose we define a metric termed the *energy overhead*. Energy overhead refers to the amount of *additional* energy that is expended in the execution of a particular implementation of a parallel or distributed simulation on some hardware configuration relative to an energy-efficient sequential execution of the same computation.

This definition is motivated by the traditional definition of speedup. Like the speedup definition, "performance" is defined relative to an efficient sequential implementation. Also like speedup, this definition is meant to encourage the development of approaches to minimizing energy consumption recognizing that a baseline amount of energy must necessarily be consumed by the computation, just as speedup recognizes that a certain amount of time is required to complete the computation on a sequential machine. Energy overhead highlights the cost of the parallel/distributed implementation in terms of energy consumption.

Just as speedup may be determined using *strong* or *weak scaling*, similar methodologies apply in measuring energy overhead. In strong scaling the speedup is computed by comparing the execution time of a fixed sized sequential computation with a parallel implementation of the same computation distributed across a parallel processor. In this light *strongly scaled energy overhead* is computed as $E_P(N) - E_S$ where E_S is the energy consumed by a sequential implementation of the computation, and $E_P(N)$ is the energy consumed by a parallel/distributed implementation of the same computation distributed over N processors.

Alternatively, speedup computed using weak scaling involves scaling the size of the computation in proportion with the number of processors. Here, *weakly scaled energy overhead* is defined as $E_{P'}(N) - E_{S'}$ where $E_{S'}$ is the energy consumed by a computation C on a sequential machine and $E_{P'}(N)$ is the energy consumed by the same computation C executing on a single processor of the parallel/distributed machine with N processors and where the entire computation executed on the parallel/distributed machine is of size $C*N$. Weakly scaled energy overhead provides insight into the amount of additional energy consumed by the parallel or distributed computation as it is scaled to larger sizes in proportion to the size of the parallel/distributed computer.

The energy overhead is impacted by several factors. It clearly depends on the hardware configuration, including consideration of memory and communications circuits, and system software on which the parallel/distributed simulation executes. Energy consumption depends on any energy-saving techniques such as DVFS that are used. Our particular concern is the overhead associated with the synchronization algorithm. As will be discussed next, energy consumption depends on the software architecture used for the implementation.

The above discussion focused on energy overhead. Similar metrics for *power* overhead can also be defined. In this context, the power overhead refers to the additional amount of power required to execute the parallel/distributed simulation relative the sequential simulation.

7.5 Distributed Simulation Systems

In this study we compare two distributed simulation middleware approaches using conservative synchronization algorithms. These two approaches utilize a peer-to-peer and a client-server architecture, respectively. The context in which we envision these architectures to be deployed might be an embedded DDDAS application where the distributed simulation executes on a power-constrained mobile computing platform, possibly connected via wireless links to a local server. This architecture places the simulations in close physical proximity to online sources of data.

The "classic" approach to implementing a parallel discrete event simulation (PDES) program is to use a *peer-to-peer architecture* where each processor or node has approximately the same computational capabilities as other nodes. The logical processes (LPs) making up the PDES program are mapped to different computation nodes using a mapping algorithm or heuristic. LPs communicate directly with other LPs by sending messages to the appropriate nodes. Early work in PDES focused almost exclusively on this approach, and to this day, this approach is often used to implement parallel or distributed simulation systems. This architecture is depicted in Figure 7.1(a) below.

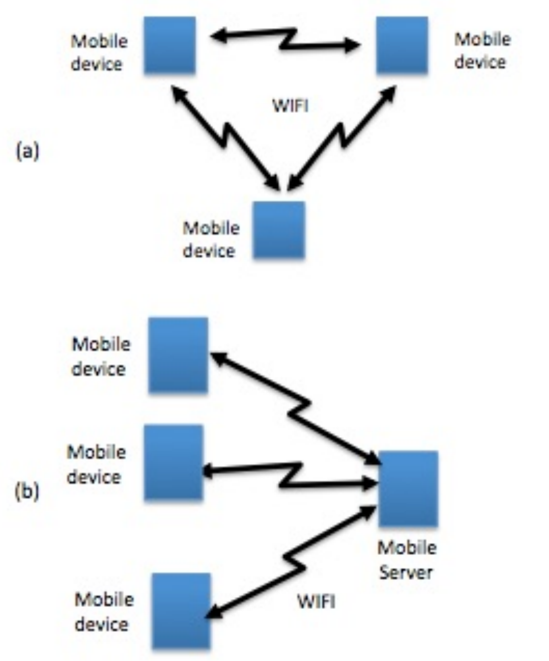


Figure 7.1. Peer-to-peer and client-server approaches.

In the peer-to-peer architecture a distributed algorithm is typically used to implement synchronization. Asynchronous algorithms use direct peer-to-peer communications between LPs (or processors). The Chandy/Misra/Bryant (CMB) algorithm is perhaps the most well known example of this approach (Chandy and Misra 1979; Bryant 1977). The principal source of energy overhead for this algorithm results from transmitting null messages between processors.

A second approach to distributed simulation is the client/server architecture, as shown in Figure 7.1(b). Logical processes execute within client processors while the simulation engine executes within the server. Here, in the context of the applications described earlier, we envision a mobile server that might reside in a special device, e.g., a larger gasoline-powered UAV. In general, however, the server might utilize the same mobile processor as client nodes in which case the distinction between clients and servers is largely logical, or the server might utilize a different, likely more powerful machine.

Clients only communicate with the server; direct client-to-client communications are not allowed. Two key functions performed by the server include forwarding messages sent between LPs residing in different clients and synchronization among the LPs/clients. This architecture is sometime used in federated distributed simulations, e.g., those based on the High Level Architecture standard, where RTI services are for the most part implemented within the server.

A natural, straightforward approach to implementing synchronization in client-server architectures are synchronous algorithms that utilize global synchronization points. The computation executes through a sequence of epochs where each involves determining those events that can be executed in this epoch with timestamps that are guaranteed to be smaller than any event that might later be received. Well known synchronous algorithms include YAWNS (Nicol 1993) and Bounded Lag (Lubachevsky 1989). A principal source of energy overhead for this algorithm lies in the barrier synchronization mechanism and lower-bound-on-timestamp (LBTS) computation that is required. Each epoch includes a barrier and computation of the LBTS value indicating the minimum timestamp of any event that might be generated in the future. More precisely, each client processor computes the smallest time stamp for any new message it might produce in the absence of receiving any additional messages as $T_i + L$ where T_i is the timestamp of the next unprocessed local event within the processor and L is the looakhead for the processor. LBTS is defined as the minimum among all of these values produced by the different processors. All events with timestamp less than LBTS are safe to process.

7.6 Experimental Configuration and Benchmark Application

The experimental configuration is intended to mimic an embedded distributed simulation application where the distributed simulation executes within a set of mobile processors. In the peer-to-peer system the mobiles make up the entire hardware platform. In the client-server architecture we posit the server resides in a location where a power source is readily available so energy use within the server is of secondary importance.

Experiments were performed to compare the CMB (peer-to-peer) and YAWNS (client-server) algorithms. A LG Nexus 5 cellular phone with a quadcore Qualcomm MSM8974 Snapdragon 800 processor, 2 GB memory, and 16 GB storage was used as the mobile computing platform. While the systems we envision may not necessarily use cellular phones as their compute engine, they most likely will utilize the same mobile processors that are used in cellular phones. The phone runs the Android version 5.0.1 (Android Lollipop) operating system and was used in the peer-to-peer experiments. The same phone was used as the client in the client-server architecture. A laptop computer was used as the server for the latter architecture. Hardware-based techniques to reduce power consumption such as voltage or frequency scaling were not used in these experiments.

All inter-processor communications utilizes wireless links. In both cases the device's 802.11n WiFi network interface was used for communications between processors. A private wireless network was established among the devices to avoid interference resulting from Internet traffic. The cellular network capability of the phone is not used in these experiments.

The energy and power consumption data are derived from direct measurement of the Android device. Specifically, the value of the instantaneous power being consumed by the device is calculated by multiplying the instantaneous battery current given by the constant integer BATTERY_PROPERTY_CURRENT_NOW and the current battery voltage level given by the constant string EXTRA_VOLTAGE. Both of these appear in the "BatteryManager" class of the "android.os" API. The instantaneous power consumption so obtained is then used to calculate the energy consumption over time.

Energy Used By Synchronization Algorithms

An initial set of experiments were conducted to measure the amount of power used by the synchronization algorithm. This was accomplished by creating benchmark programs where each LP only processed local events, and did not exchange events with other processors. This ensures that interprocessor communication is only utilized by the synchronization algorithm rather than passing event messages. In these experiments each LP is initialized with some number of local events, and processing each event causes one new locally scheduled event to be scheduled with a fixed time stamp increment. A set of experiments were then performed using the CMB and YAWNS implementations as lookahead was varied. Because no events are scheduled between processors, the lookahead could be set to arbitrary values without concern of violating lookahead constraints. In these experiments the simulation benchmark program is the same across all lookahead values and both synchronization algorithms, enabling fair comparisons.

The amount of energy consumed by the benchmark programs for different lookahead values are shown in Figure 7.2. In this figure both lookahead and energy are plotted on logarithmic scales. It is seen that lookahead can have a dramatic effect on the amount of power consumed by the synchronization algorithm – two orders of magnitude across the lookahead values used in these experiments.

The CMB algorithm yielded energy consumption that steadily decreased as the lookahead was increased. For very small lookahead values CMB is prone to a phenomena called lookahead creep where null messages must be sent among the processors to, in effect, enable them to advance by an amount of simulation time equal to the lookahead. To a first-order approximation, doubling the lookahead value approximately doubles the amount of time advance that can be gained with each "round" of null messages. Assuming the primary cause of energy utilization is the time to send null messages, the data shown in Figure 7.1 is consistent with this observation where it is seen a steady decline in energy consumption results as the lookahead value is increased.

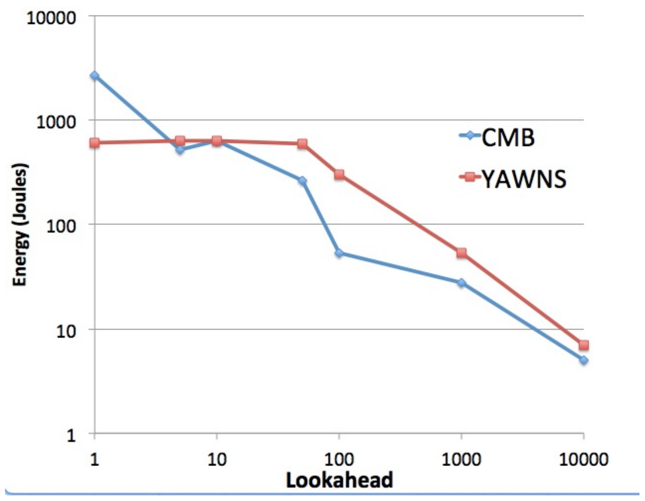


Figure 7.2: Energy consumed for Chandy/Misra/Bryant and YAWNS synchronization algorithms as lookahead is varied.

The YAWNS experiments yielded a decidely different behavior. Here, the energy consumptions remains at a relatively constant level for small to moderate lookahead values. However, energy consumption then steadily decreases with lookahead increases at relatively high lookahead values. In contrast to CMB, YAWNS is not prone to the lookahead creep problem in the sense that the algorithm exploits knowledge of the timestamp of the next unprocessed event to advance simulation time. Consider the case where the lookahead is very small, say 1, and the average time between events is 10 units of simulation time. In accordance with time creep, CMB must advance each LP by increments of 1 with each round of null messages to advance LPs to the point where they can process the next (non-null) event. On the other hand, YAWNS will immediately advance the LP to the time of the next event. If the lookahead is small, YAWNS will, again to a first-order estimate, advance LBTS to the timestamp of the next unprocessed simulation event. Therefore the energy required for synchronization is in proportion to the number of events, independent of the lookahead value, explaining why energy consumption remains flat for small lookahead values. This remains true until the lookahead becomes large. With large lookahead values, many events can be processed in each epoch of YAWNS. Roughly speaking, doubling the lookahead value approximately doubles the number of events that can be processed within each epoch. Thus, for large lookahead values, the energy overhead steadily declines as the lookahead increases. The data in Figure 7.1 is consistent with this explanation.

Overall, it can be seen that YAWNS and CMB use roughly comparable amounts of energy in the experiments with large lookahead values. However, these measuements also suggest CMB expends considerably more energy at very low lookahead values due to the lookahead creep problem.

Queueing Network Simulations

A second benchmark program used in this study is a simulation of a closed queueing network with *J* jobs circulating among the nodes of the network. The queueing network is configured as a three-dimensional toroid topology. Each processor is assigned one two-dimensional plane of the toroid. Once a job receives service it is routed to a randomly selected neighboring node, with each neighbor equally likely to be selected. Each node of the network contains a single server with service time drawn from an exponential distribution plus a constant value *L*. Jobs arriving at each network node are placed into a single queue, and are served in first-come-first-serve order. The minimum service time *L* is used as a control variable to facilitate experimentation with increased lookahead values. In these experiments the lookahead is enhanced by pre-sampling the random number generator to produce the service time of the next job to be processed by the server; if the pre-sampled value is *P*, then the time stamp of the next message generated by the LP must be at least *L+P* units of simulation time into the future (Nicol 1988). Further, the simulation is optimized to exploit the fact that the queue uses a FCFS queueing discipline, resulting in increased lookahead in proportion to the queueing delay. The benchmark program is written in C.

Figure 7.3 shows the energy consumed by the three simulations for queueing networks of size 4 (2x2), 49 (7x7), 484 (22x22), and 1024 (32x32) nodes executing on each processor. The total number of events processed by the simulators was kept constant across all of these runs, i.e., the total amount of simulation computation remained the same across the runs. This figure shows energy consumption data; power consumption data demonstrated similar trends to that shown in the figure.

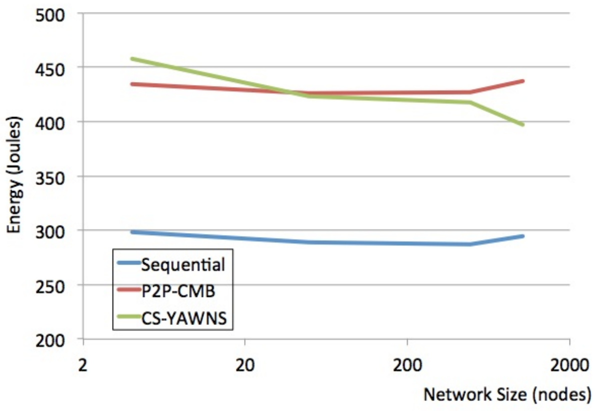


Figure 7.3. Energy consumption for 2x2, 7x7, 22x22, and 32x32 queueing networks. Note network size is plotted on a logarithmic scale.

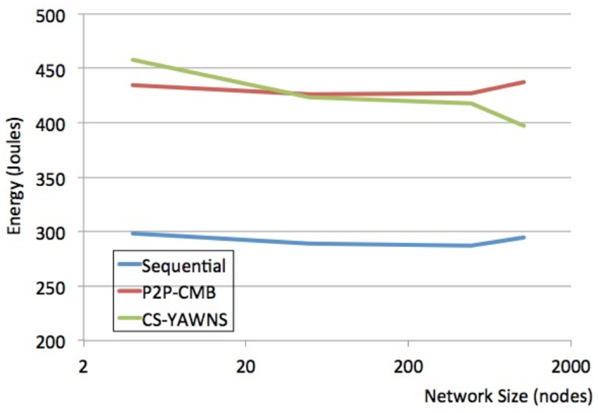


Figure. 7.3: Energy consumption for 2x2, 7x7, 22x22, and 32x32 queueing networks. Note network size is plotted on a logarithmic scale.

It can be seen that the energy consumed by the sequential and P2P-CMB simulations remains about constant as network size is increased, but there is a modest reduction of approximately 13% in energy consumed by the CS-YAWNS simulation for the largest network compared to the smallest. We believe this is due to a much smaller number of synchronization messages in CS-YAWNS in the larger sized queueing networks. As discussed earlier YAWNS operates using a scheme where all events in the current epoch can be safely processed without concern for events later arriving with a smaller timestamp. For these experiments the lookahead, i.e., the minimum timestamp increment, remains the same across all runs. Therefore, there will be more events within a single epoch that may be processed before the next global synchronization. Since the total number of events in the computation remains the same, this results in fewer global synchronization points. The number of synchronization messages in YAWNS was reduced in approximately the same proportion as the size of the network (a factor of 256) across these experiments.

Varying the size of the network changes the amount of computation performed between communications. Larger networks will have more simulation computations to complete between successive interprocessor message communications, likely accounting for the differences shown in this figure.

The energy overhead resulting from the distributed execution of the simulation program relative the sequential implementation using these two synchronization algorithms and architectures is shown in Figure 7.4. These data are derived by subtracting the energy consumed by the sequential implementation from the distributed execution for each network size, and plotting the resulting value as a percentage of the energy expended in the sequential execution. As can be seen, the distributed simulations expend from 35% to 54% more energy than the sequential simulation executing the same number of events. While the P2P implementation of CMB remains approximately the same percentage of energy overhead for the different sized networks, there is more variability in the client-server YAWNS implementation.

7.7 Discussion

One can observe a few trends that emerge across these experiments. First, these data highlight the energy cost resulting from the distributed execution of a simulation program is significant. Without more detailed measurements of the architecture itself, one cannot definitively pinpoint all of the causes of this increased energy usage, however it seems clear that interprocessor communication for event passing and synchronization is most likely a principal factor. These measurements indicate that the energy overhead is significant for these conservative synchronization algorithms.

Second, we observe that different synchronization algorithms and architectures exhibit different behaviors with respect to energy consumption. It is clear that different synchronization algorithms will exhibit different message passing behaviors, so in this sense it is not surprising that they yield different energy consumption characteristics. These data indicate that these differences can be significant, and can lead to observable differences in energy consumption. Thus, for applications where energy is a critical factor, e.g., for distributed simulations executing on mobile devices, some care should be taken with respect to the choice of synchronization algorithm in order to maximize battery life.

Third, aspects of the distributed simulation application such as lookahead that impact the behavior of the synchronization algorithm may have a significant impact on the energy efficiency of the distributed simulation. Further investigation is required to examine the impact of aspects such as lookahead on energy efficiency and to gain a deep understanding of this relationship.

When comparing the energy consumption of these two, very different, synchronization algorithms, two observations are apparent. First, to a first order approximation, the overall, average energy overhead observed for these synchronization algorithms is significant, and to some extent, relatively similar. One the other hand, the two algorithms exhibit different energy characteristics as parameters of the simulation such as the number of LPs and lookahead change. One must be cautionary in that these observations are based on a very limited amount of experimental data. Nevertheless, these results suggest that further exploration of the relationship between the synchronization algorithm and energy and power consumption is warranted.

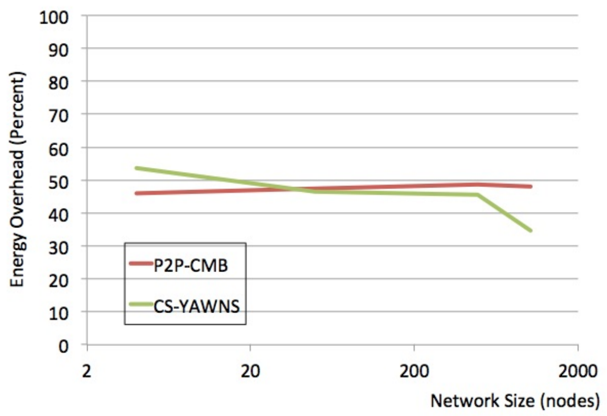


Figure 7.4: Energy overhead of CMB and YAWNS for different sized queueing networks as a percentage of the energy expended by the sequential execution.

7.8 Conclusions and Future Work

We have argued that power and energy consumption are areas of increasing concern for parallel and distributed simulation systems. In contexts such as embedded and mobile systems and some high performance computing applications we believe these aspects are of sufficient importance to merit explicit consideration in the design of the system.

Metrics are proposed focusing on the energy and power efficiency of parallel and distributed simulations relative to a sequential simulation. The metrics focused on developing measures that are consistent and complementary to standard metrics used for some time, specifically speedup under strong and weak scaling assumptions.

The empirical work presented here represents an initial evaluation of the energy and power consumed for parallel and distributed simulations, focusing on the synchronization algorithm that is used. Experimental measurements of operational distributed simulation systems demonstrate that distributed execution incurs a significant overhead in energy consumption to execute the simulation. Clearly the amount of this overhead will depend on the application, but queueing networks, a standard benchmark used for parallel and distributed simulation, of various sizes all demonstrate significant energy overhead. Architectural choices and the synchronization algorithm can lead to different results concerning the amount of energy and power that is consumed. In these experiments the client-server YAWNS implementation and the Chandy/Misra/Bryant algorithm executing on a peer-to-peer architecture using WiFi communications yielded different behaviors with respect ti energy overheads. The two approaches also exhibited different detailed behaviors for different queueing network sizes.

Power- and energy- consumption of parallel and distributed simulations represents a new area of research that has many unanswered questions. Deep understandings of the power and energy consumed by distributed simulations, and those aspects that are different relative to other types of computing applications do not yet exist. Accurate, predictive models of energy consumption of parallel and distributed simulations taking into account the machine architecture as well as the behavior of the simulation application are needed. A comprehensive examination of alternate conservative synchronization algorithms and their execution on different distributed system architectures across a wide variety of parameter settings is required. Similar evaluations and comparisons with optimistic synchronization techniques are needed. Energy efficiency on platforms such as cloud and grid computing environments requires further study. Extensive analyses for real world applications are needed both for embedded and mobile simulations as well as high performance computing systems.

Beyond analysis studies, the development and evaluation of techniques to reduce energy- and power- consumption of distributed simulations is an unexplored area of research. We refer to techniques that take into consideration power and energy consumption in the design and operation of the system energy- and power-aware distributed simulations. One can envision dynamically changing the behavior of the distributed simulation based on the energy available in batteries, or based on the power being consumed on a high performance computing platform. Distributed simulations executing in a real-time context such as those that arise in DDDAS impose time constraints to completing the computation. Moreover, we believe power- and energy-aware parallel and distributed simulation represents a rich area of future research for the field with many unsolved problems.

8 A High-Performance Parallel Algorithm for Nonnegative Matrix Factorization

We now shift our attention to the sensing portion of the DDDAS processing loop. Specifically, the proposed DDAS approach relies on detection and re-detection of vehicles. Image processing algorithms are required for this purpose. Here, we explore the use of non-negative matrix factorization (NMF) to detect vehicles from captured video streams. Because real-time or near real-time performance is needed, the performance of NMF algorithms is an important issue.

Non-negative matrix factorization is the problem of determining two non-negative low rank factors **W** and **H**, for the given input matrix **A**, such that **A**≈**WH**. NMF is a useful tool for many applications in different domains such as topic modeling in text mining, background separation in video analysis, and community detection in social networks. Despite its popularity in the data mining community, there is a lack of efficient parallel software to solve the problem for big datasets. Existing distributed-memory algorithms are limited in terms of performance and applicability, as they are implemented using Hadoop and are designed only for sparse matrices.

We propose a distributed-memory parallel algorithm that computes the factorization by iteratively solving alternating non-negative least squares (NLS) subproblems for **W** and **H**. To our knowledge, our algorithm is the first high-performance parallel algorithm for NMF. It maintains the data and factor matrices in memory (distributed across processors), uses MPI for interprocessor communication, and, in the dense case, provably minimizes communication costs (under mild assumptions). As opposed to previous implementations, our algorithm is also flexible: (1) it performs well for dense and sparse matrices, and (2) it allows the user to choose from among multiple algorithms for solving local NLS subproblems within the alternating iterations. We demonstrate the scalability of our algorithm and compare it with baseline implementations, showing significant performance improvements.

8.1 Introduction

Non-negative Matrix Factorization (NMF) is the problem of finding two low rank factors $\mathbf{W} \in \mathbb{R}_+^{m\times k}$ and $\mathbf{H} \in \mathbb{R}_+^{k\times n}$ for a given input matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, such that $\mathbf{A} \approx \mathbf{WH}$, where $k \ll min(m,n)$. Formally, NMF problem [17] can be defined as

$$\min_{\mathbf{W}\geqslant 0, \mathbf{H}\geqslant 0} f(\mathbf{W},\mathbf{H}) \equiv \|\mathbf{A}-\mathbf{WH}\|_F^2. \qquad (1)$$

NMF is widely used in data mining and machine learning as a dimension reduction and factor analysis method. It is a natural fit for many real world problems as the non-negativity is inherent in many representations of real-world data and the resulting low rank factors are expected to have natural interpretation. The applications of NMF range from text mining (Pauca et al. 2004), computer vision (Hoyer 2004), bioinformatics (Kim and Park 2007), to blind source separation Cichocki et al. 2009), unsupervised clustering (Kuang et al. 2012) and many other areas. For typical problems today, *m* and *n* can be on the order of thousands to millions or more, and *k* is typically less than 100.

There is a vast literature on algorithms for NMF and their convergence properties (Kim et al. 2014). The commonly adopted NMF algorithms are -- (i) Multiplicative Update (MU) (Seung and Lee 2001) (ii) Hierarchical Alternative Least Squares (HALS) (Cichocki et al. 2009) (iii) Block Principal Pivoting (BPP) (Kim and Park 2011) and (iv) Stochastic Gradient Descent (SGD) Updates (Gemulla et al. 2011). As described in Equation 2, most of the algorithms in NMF literature are based on Alternating Non-negative Least Squares (ANLS) scheme that iteratively optimizes each of the low rank factors **W** and **H** while keeping the other fixed. It is important to note that in such iterative alternating minimization techniques, each subproblem is a constrained convex optimization problem. Each of these subproblems is then solved using standard optimization techniques such as projected gradient, interior point, etc., and a detailed survey for solving this constrained convex optimization problem can be found in (Kim et al. 2014; Wang and Zhang 2013). In this chapter, for solving the subproblems, our implementation uses a fast active-set based method called Block Principal Pivoting (BPP) (Kim and Park 2007), but the parallel algorithm proposed in this chapter can be easily extended for other algorithms such as MU and HALS.

Recently with the advent of large scale internet data and interest in Big Data, researchers have started studying scalability of many foundational machine learning algorithms. To illustrate the dimension of matrices commonly used in the machine learning community, we present a few examples. Now-a-days the adjacency matrix of a billion-node social network is common. In the matrix representation of a video data, every frame contains three matrices for each RGB color, which is reshaped into a column. Thus in the case of a 4K video, every frame will take approximately 27 million rows (4096 row pixels x 2196 column pixels x 3 colors). Similarly, the popular representation of documents in text mining is a bag-of-words matrix, where the rows are the dictionary and the columns are the documents (e.g., webpages). Each entry A_{ij} in the bag-of-words matrix is generally the frequency count of the word *i* in the document *j*. Typically with the explosion of the new terms in social media, the number of words spans to millions.

To handle such high dimensional matrices, it is important to study low rank approximation methods in a data-distributed environment. For example, in many large scale scenarios, data samples are collected and stored over many general purpose computers, as the set of samples is too large to store on a single machine. In this case, the computation must also be distributed across processors. Local computation is preferred as local access of data is much faster than remote access due to the costs of interprocessor communication. However, for low rank approximation algorithms that depend on global information (like MU, HALS, and BPP), some communication is necessary.

The simplest way to organize these distributed computations on large datasets is through a MapReduce framework like Hadoop, but this simplicity comes at the expense of performance. In particular, most MapReduce frameworks require data to be read from and written to disk at every iteration, and they involve communication-intensive global, input-data shuffles across machines.

In this work, we present a much more efficient algorithm and implementation using tools from the field of High-Performance Computing (HPC). We maintain data in memory (distributed across processors), take advantage of optimized libraries for local computational routines, and use the Message Passing Interface (MPI) standard to organize interprocessor communication. The current trend for high-performance computers is that available parallelism (and therefore aggregate computational rate) is increasing much more quickly than improvements in network bandwidth and latency. This trend implies that the relative cost of communication (compared to computation) is increasing.

To address this challenge, we analyze algorithms in terms of both their computation and communication costs. In particular, our proposed algorithm ensures that after the input data is initially read into memory, it is *never* communicated; we communicate only the factor matrices and other smaller temporary matrices. Furthermore, we prove that in the case of dense input and under the assumption that $k \leqslant \sqrt{mn/p}$, our proposed algorithm *minimizes* bandwidth cost (the amount of data communicated between processors) to within a constant factor of the lower bound. We also reduce latency costs (the number of times processors communicate with each other) by utilizing MPI collective communication operations, along with temporary local memory space, performing O(log p) messages per iteration, the minimum achievable for aggregating global data.

Fairbanks et al. (2015) discuss a parallel NMF algorithm designed for multicore machines. To demonstrate the importance of minimizing communication, we consider this approach to parallelizing an alternating NMF algorithm in distributed memory. While this naive algorithm exploits the natural parallelism available within the alternating iterations (the fact that rows of **W** and columns of **H** can be computed independently), it performs more communication than necessary to set up the independent problems. We compare the performance of this algorithm with our proposed approach to demonstrate the importance of designing algorithms to minimize communication; that is, simply parallelizing the computation is not sufficient for satisfactory performance and parallel scalability.

The main contribution of this work is a new, high-performance parallel algorithm for non-negative matrix factorization. The algorithm is flexible, as it is designed for both sparse and dense input matrices and can leverage many different algorithms for solving local non-negative least squares problems. The algorithm is also efficient, maintaining data in memory, using MPI collectives for interprocessor communication, and using efficient libraries for local computation. Furthermore, we provide a theoretical communication cost analysis to show that our algorithm reduces communication relative to the naive approach, and in the case of dense input, that it provably minimizes communication. We show with performance experiments that the algorithm outperforms the naive approach by significant factors, and that it scales well for up to 100s of processors on both synthetic and real-world data.

8.2 Preliminaries

Notation

Table 1 summarizes the notation we use throughout. We use *upper case* letters for matrices and *lower case* letters for vectors. For example, **A** is a matrix and **a** is a column vector and $\mathbf{a}^T$ is a row vector. The subscripts are used for sub-blocks of matrices. We use *m* and *n* to denote the numbers of rows and columns of **A**, respectively, and we assume without loss of generality *m* > *n* throughout.

$\mathbf{A}$	Input Matrix
$\mathbf{W}$	Left Low Rank Factor
$\mathbf{H}$	Right Low Rank Factor
m	Number of rows of input matrix
n	Number of columns of input matrix
k	Low Rank
$\mathbf{M}_i$	ith row block of matrix $\mathbf{M}$
$\mathbf{M}^i$	ith column block of matrix $\mathbf{M}$
$\mathbf{M}_{ij}$	(i, j)th subblock of $\mathbf{M}$
p	Number of parallel processes
p_r	Number of rows in processor grid
p_c	Number of columns in processor grid

Table 8.1: Notation

Communication Model

To analyze our algorithms, we use the *α-β-γ* model of distributed-memory parallel computation. In this model, interprocessor communication occurs in the form of messages sent between two processors across a bidirectional link (we assume a fully connected network). We model the cost of a message of size n words as $\alpha + n\times\beta$, where α is the per-message latency cost and β is the per-word bandwidth cost. Each processor can compute floating point operations (flops) on data that resides in its local memory; γ is the per-flop computation cost. With this communication model, we can predict the performance of an algorithm in terms of the number of flops it performs as well as the number of words and messages it communicates. For simplicity, we will ignore the possibilities of overlapping computation with communication in our analysis. For more details on the *α-β-γ* model, see (Chan et al. 2007; Thakur et al. 2005).

MPI collectives

Point-to-point messages can be organized into collective communication operations that involve more than two processors. MPI provides an interface to the most commonly used collectives like broadcast, reduce, and gather, as the algorithms for these collectives can be optimized for particular network topologies and processor characteristics. The algorithms we consider use the all-gather, reduce-scatter, and all-reduce collectives, so we review them here, along with their costs. Our analysis assumes optimal collective algorithms are used, see (Cichocki et al. 2009; Thakur et al. 2005), though our implementation relies on the underlying MPI implementation.

At the start of an all-gather collective, each of p processors owns data of size n/p. After the all-gather, each processor owns a copy of the entire data of size n. The cost of an all-gather is $\alpha \cdot \log p + \beta \cdot \frac{p-1}{p} n$ At the start of a reduce-scatter collective, each processor owns data of size n. After the reduce-scatter, each processor owns a subset of the sum over all data, which is of size n/p. (Note that the reduction can be computed with other associative operators besides addition.) The cost of an reduce-scatter is $\alpha \cdot \log p + (\beta + \gamma) \cdot \frac{p-1}{p} n$ At the start of an all-reduce collective, each processor owns data of size n. After the all-reduce, each processor owns a copy of the sum over all data, which is also of size n. The cost of an all-reduce is $2\alpha \cdot \log p + (2\beta + \gamma) \cdot \frac{p-1}{p} n$ Note that the costs of each of the collectives are zero when p=1.

8.3 Related Work

In the data mining and machine learning literature there is an overlap between low rank approximations and matrix factorizations due to the nature of applications. Despite its name, Non-negative ``Matrix Factorization" is really a low rank approximation. In this section, we discuss the various distributed NMF algorithms.

The recent distributed NMF algorithms in the literature are (Liao et al. 2014; Gemulla et al. 2011; Yin et al. 2014; Faloutsos et al. 2014). All of these works propose distributed NMF algorithms implemented using Hadoop. Liu et al. (2010) propose running Multiplicative Update (MU) for KL divergence, squared loss, and ``exponential" loss functions. Matrix multiplication, element-wise multiplication, and element-wise division are the building blocks of the MU algorithm. The authors discuss performing these matrix operations effectively in Hadoop for sparse matrices. In similar directions, Liao et al. (2014) implement an open source Hadoop based MU algorithm and study its scalability on large-scale biological datasets. Also, Yin et al. (2014) present a scalable NMF that can perform frequent updates, which aim to use the most recently updated data. Gemmula et al. (2011) propose a *generic algorithm* that works on different loss functions, often involving the distributed computation of the gradient. According to the authors, this formulation can also be extended to handle non-negative constraints. Similarly Faloutsos et al. (2014) propose a distributed, scalable method for decomposing matrices, tensors, and coupled data sets through stochastic gradient descent on a variety of objective functions. The authors also provide an implementation that can enforce non-negative constraints on the factor matrices.

8.4 Foundations

In this section, we will briefly introduce the Alternating Non-negative Least Squares (ANLS) framework, multiple methods for solving non-negative least squares problems (NLS) including Block Principal Pivoting (BPP), and a straightforward approach to parallelization of the framework.

Alternating Non-negative Least Squares

According to the ANLS framework, we first partition the variables of the NMF problem into two blocks **W** and **H**. Then we solve the following equations iteratively until a stopping criteria is satisfied.

$$\begin{aligned} \mathbf{W} &\leftarrow \underset{\tilde{\mathbf{W}}\geqslant 0}{\operatorname{argmin}} \left\|\mathbf{A} - \tilde{\mathbf{W}}\mathbf{H}\right\|_F^2, \\ \mathbf{H} &\leftarrow \underset{\tilde{\mathbf{H}}\geqslant 0}{\operatorname{argmin}} \left\|\mathbf{A} - \mathbf{W}\tilde{\mathbf{H}}\right\|_F^2. \end{aligned} \tag{2}$$

The optimizations sub-problem for **W** and **H** are NLS problems which can be solved by a number of methods from generic constrained convex optimization to active-set methods. Typical approaches use form the normal equations of the least squares problem (and then solve them enforcing the non-negativity constraint), which involves matrix multiplications of the factor matrices with the data matrix.Algorithm 1 shows this generic approach.

Algorithm 1 [**W**, **H**] = ANLS(A, k)

```
Require: A is an $m \times n$ matrix, $k$ is rank of approximation
1: Initialize H with a non-negative matrix in $\mathbb{R}_+^{n\times k}$.
2: while not converged do
3:     Update W using $\mathbf{HH}^T$ and $\mathbf{AH}^T$
4:     Update H using $\mathbf{W}^T\mathbf{W}$ and $\mathbf{W}^T\mathbf{A}$
5: end while
```

The specifics of lines 3 and 4depend on the NLS method used. For example, the update equations for MU [17] are

$$\begin{aligned} w_{ij} &\leftarrow w_{ij}\frac{(\mathbf{AH}^T)_{ij}}{(\mathbf{WHH}^T)_{ij}} \\ h_{ij} &\leftarrow h_{ij}\frac{(\mathbf{W}^T\mathbf{A})_{ij}}{(\mathbf{W}^T\mathbf{WH})_{ij}}. \end{aligned} \tag{3}$$

Note that after computing $\mathbf{HH}^T$ and $\mathbf{AH}^T$, the cost of updating **W** is dominated by computing $\mathbf{WHH}^T$, which is $2mk^2$ flops. Given $\mathbf{AH}^T$ and $\mathbf{WHH}^T$, each entry of **W** can be updated independently and cheaply. Likewise, the extra computation cost of updating **H** is $2nk^2$ flops.

HALS updates **W** and **H** by applying block coordinate descent on the columns of **W** and rows of **H** [3].The update rules are

$$\begin{aligned} \mathbf{w}^i &\leftarrow \frac{\left[(\mathbf{AH}^T)^i - \sum_{\substack{l=1\\l\neq i}}^{k}(\mathbf{HH}^T)_{li}\mathbf{w}^l\right]_+}{(\mathbf{HH}^T)_{ii}} \\ \mathbf{h}_i &\leftarrow \frac{\left[(\mathbf{W}^T\mathbf{A})_i - \sum_{\substack{l=1\\l\neq i}}^{k}(\mathbf{W}^T\mathbf{W})_{li}\mathbf{h}_l\right]_+}{(\mathbf{W}^T\mathbf{W})_{ii}}, \end{aligned} \tag{4}$$

where w^i is the ith column of **W** and h_i is the i row of **H**. Note that the columns of **W** and rows of **H** are updated in order, so that the most up-to-date values are used in the update.

The extra computation is again $2(m+n)k^2$ flops for updating both **W** and **H**.

Block Principal Pivoting

In this chapter, we focus on and use the block principal pivoting (Kim and Park 2011) method to solve the non-negative least squares problem, as it has the fastest convergence rate (in terms of number of iterations).

We note that any of the NLS methods can be used within our parallel frameworks (Algorithms 2 and 3).

BPP is the state-of-the-art method for solving the NLS subproblems in Eq. 2. The main sub-routine of BPP is the single right-hand side NLS problem

$$\min_{\mathbf{x}\geqslant 0} \|\mathbf{C}\mathbf{x}-\mathbf{b}\|_2^2. \tag{5}$$

The Karush-Kuhn-Tucker (KKT) optimality condition for Eq. 5 is as follows

$$\mathbf{y} = \mathbf{C}^T\mathbf{C}\mathbf{x} - \mathbf{C}^T\mathbf{b} \tag{6a}$$

$$\mathbf{y} \geqslant 0 \tag{6b}$$

$$\mathbf{x} \geqslant 0 \tag{6c}$$

$$\mathbf{x}^T\mathbf{y} = 0. \tag{6d}$$

The KKT condition (6) states that at optimality, the support sets (i.e., the non-zero elements) of **x** and **y** are complementary to each other. Therefore, Eq. (6) is an instance of the *Linear Complementarity Problem* (LCP) which arises frequently in quadratic programming. When k << min(*m, n*), active set and active-set like methods are very suitable because most computations involve matrices of sizes $m \times k, n \times k,$ and $k \times k$ which are small and easy to handle.

If we knew which indices correspond to nonzero values in the optimal solution, then computing it is an unconstrained least squares problem on these indices. In the optimal solution, call the set of indices *i* such that $x_i=0$ the active set, and let the remaining indices be the passive set. The BPP algorithm works to find this active set and passive set. Since the above problem is convex, the correct partition of the optimal solution will satisfy the KKT condition (Eq. 6). The BPP algorithm greedily swaps indices between the active and passive set until finding a partition that satisfies the KKT condition. In the partition of the optimal solution, the values of the indices that belong to the active set will take zero. The values of the indices that belong to the passive set are determined by solving the least squares problem using normal equations restricted to the passive set. Kim et al. (2011) discuss the BPP algorithm in further detail. Because the algorithm is iterative, we define $C_{\text{BPP}}(k, c)$ as the cost of solving *c* instances of Eq. 5 using BPP, given the k×k matrix $\mathbf{C}^T\mathbf{C}$ and *c* columns of the form **b**.

Naive Parallel NMF Algorithm

In this section we present a naive parallelization of any NMF algorithm (Fairbanks 2015) that follows the ANLS framework (Algorithm 1). Each NLS problem with multiple right-hand sides can be parallelized on the observation that the problems for multiple right-hand sides are independent from each other. That is, we can solve several instances of Eq. 5 independently for different **b** where **C** is fixed, which implies that we can optimize row blocks of **W** and column blocks of **H** in parallel.

Algorithm 2 presents a straightforward approach to setting up the independent subproblems. Let us divide **W** into row blocks $\mathbf{W}_1,...,\mathbf{W}_p$ and **H** into column blocks $\mathbf{H}^1,..., \mathbf{H}^p$. We then double-partition the data matrix **A** accordingly into row blocks $\mathbf{A}_1,..., \mathbf{A}_p$ and column blocks $\mathbf{A}^1,..., \mathbf{A}^p$ so that processor *i* owns both $\mathbf{A}_i$ and $\mathbf{A}^i$ (see Figure 8.1). With these partitions of the data and the variables, one can implement any ANLS algorithm in parallel, with only one communication step for each solve.

```
Algorithm 2 $[\mathbf{W}, \mathbf{H}]$ = Naive-Parallel-NMF($\mathbf{A}, k$)
Require: $\mathbf{A}$ is an $m \times n$ matrix distributed both row-wise and
    column-wise across $p$ processors, $k$ is rank of approximation
Require: Local matrices: $\mathbf{A}_i$ is $m/p \times n$, $\mathbf{A}^i$ is $m \times n/p$, $\mathbf{W}_i$ is $m/p \times k$,
    $\mathbf{H}^i$ is $k \times n/p$
 1: $p_i$ initializes $\mathbf{H}^i$
 2: while not converged do
    /* Compute W given H */
 3:     collect $\mathbf{H}$ on each processor using all-gather
 4:     $p_i$ computes $\mathbf{W}_i \leftarrow \underset{\tilde{\mathbf{W}} \geqslant 0}{\operatorname{argmin}} \|A_i - \tilde{\mathbf{W}}\mathbf{H}\|$
    /* Compute H given W */
 5:     collect $\mathbf{W}$ on each processor using all-gather
 6:     $p_i$ computes $\mathbf{H}^i \leftarrow \underset{\tilde{\mathbf{H}} \geqslant 0}{\operatorname{argmin}} \|A^i - \mathbf{W}\tilde{\mathbf{H}}\|$
 7: end while
Ensure: $\mathbf{W}, \mathbf{H} \approx \underset{\tilde{\mathbf{W}} \geqslant 0, \tilde{\mathbf{H}} \geqslant 0}{\operatorname{argmin}} \|\mathbf{A} - \tilde{\mathbf{W}}\tilde{\mathbf{H}}\|$
Ensure: $\mathbf{W}$ is an $m \times k$ matrix distributed row-wise across pro-
    cessors, $\mathbf{H}$ is a $k \times n$ matrix distributed column-wise across
    processors
```

The computation cost of Algorithm 2 depends on the local NLS algorithm. For comparison with our proposed algorithm, we assume each processor uses BPP to solve the local NLS problems. Thus, the computation at line 4 consists of computing $\mathbf{A}^i\mathbf{H}^T$, $\mathbf{HH}^T$, and solving NLS given the normal equations formulation of rank k for m/p columns. Likewise, the computation at line 6 consists of computing W^TA, W^TW, and solving NLS for n/p columns. In the dense case, this amounts to $4mnk/p + (m+n)k^2 + C_{\text{BPP}}((m+n)/p, k)$ flops. In the sparse case, processor i performs $2(\text{nnz}(\mathbf{A}_i) + \text{nnz}(\mathbf{A}^i))k$ flops to compute $\mathbf{A^iH^T}$ and $\mathbf{W^TA_i}$ instead of $4mnk/p$.

The communication cost of the all-gathers at lines 3 and 5, based on the expression given in Section 2.3 is α*2log p + β*$(m+n)k$. The local memory requirement includes storing each processor's part of matrices **A**, **W**, and **H**. In the case of dense **A**, this is $2mn/p+(m+n)k/p$ words, as **A** is stored twice; in the sparse case, processor i requires nnz(A_i)+\nnz(A^i)$ words for the input matrix and $(m+n)k/p$ words for the output factor matrices. Local memory is also required for storing temporary matrices **W** and **H** of size $(m+n)k$ words.

We summarize the algorithmic costs of Algorithm 2 in Table 2. This naive algorithm (Fairbanks et al. 2015) has three main drawbacks: (1) it requires storing two copies of the data matrix (one in row distribution and one in column distribution) and both full factor matrices locally, (2) it does not parallelize the computation of $\mathbf{HH}^T$ and $\mathbf{W^TW}$ (each processor computes it redundantly), and (3) as we will see in Section 5 it communicates more data than necessary.

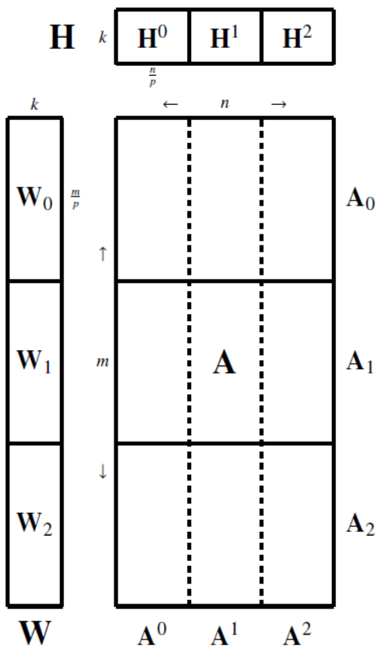


Figure 8.1: Distribution of matrices for Naive (Algorithm 2), for p=3. Note that $\mathbf{A}_i$ is $m/p*n$, $\mathbf{A}^i$ is $m*n/p$, $\mathbf{W}_i$ is $m/p_r * k$, and $\mathbf{H}^i$ is $k* n/p$.

8.5 High Performance Parallel NMF

We present our proposed algorithm, HPC-NMF, as Algorithm 3. The algorithm assumes a 2D distribution of the data matrix **A** across a $p_r \times p_c$ grid of processors (with $p=p_r p_c$), as shown in Figure 8.2. Algorithm 3 performs an alternating method in parallel with a per-iteration bandwidth cost of $O\left(\min\left\{\sqrt{mnk^2/p}, nk\right\}\right)$ words, latency cost of O(log p) messages, and load-balanced computation (up to the sparsity pattern of **A** and convergence rates of local BPP computations). To minimize the communication cost and local memory requirements, p_r and p_c are chosen so that $m/p_r \approx n/p_c \approx \sqrt{mn/p}$, in which case the bandwidth cost is $O\left(\sqrt{mnk^2/p}\right)$.

If the matrix is very tall and skinny, i.e., $m/p>n$, then we choose p_r=p and p_c=1. In this case, the distribution of the data matrix is 1D, and the bandwidth cost is O(nk) words.

The matrix distributions for Algorithm 3 are given in Figure 8.2; we use a 2D distribution of **A** and 1D distributions of **W** and **H**. Recall from Table 1 that $\mathbf{M}_i$ and $\mathbf{M}^i$ denote row and column blocks of **M**, respectively. Thus, the notation $(\mathbf{W}i)_j$ denotes the jth row block within the ith row block of **W**. Lines 3-8 compute **W** for a fixed **H**, and lines 9-14 compute **H** for a fixed **W**; note that the computations and communication patterns for the two alternating iterations are analogous.

Algorithm 3 $[\mathbf{W}, \mathbf{H}] = \text{HPC-NMF}(\mathbf{A}, k)$

```
Require: $\mathbf{A}$ is an $m \times n$ matrix distributed across a $p_r \times p_c$ grid of processors, $k$ is rank of approximation
Require: Local matrices: $\mathbf{A}_{ij}$ is $m/p_r \times n/p_c$, $\mathbf{W}_i$ is $m/p_r \times k$, $(\mathbf{W}_i)_j$ is $m/p \times k$, $\mathbf{H}_j$ is $k \times n/p_c$, and $(\mathbf{H}_j)_i$ is $k \times n/p$
 1: $p_{ij}$ initializes $(\mathbf{H}_j)_i$
 2: while not converged do
    /* Compute W given H */
 3:     $p_{ij}$ computes $\mathbf{U}_{ij} = (\mathbf{H}_j)_i(\mathbf{H}_j)_i^T$
 4:     compute $\mathbf{HH}^T = \sum_{i,j} \mathbf{U}_{ij}$ using all-reduce across all procs
                                        ▷ $\mathbf{HH}^T$ is $k \times k$ and symmetric
 5:     $p_{ij}$ collects $\mathbf{H}_j$ using all-gather across proc columns
 6:     $p_{ij}$ computes $\mathbf{V}_{ij} = \mathbf{A}_{ij}\mathbf{H}_j^T$
                                        ▷ $\mathbf{V}_{ij}$ is $m/p_r \times k$
 7:     compute $(\mathbf{AH}^T)_i = \sum_j \mathbf{V}_{ij}$ using reduce-scatter across proc row to achieve row-wise distribution of $(\mathbf{AH}^T)_i$
                                        ▷ $p_{ij}$ owns $m/p \times k$ submatrix $((\mathbf{AH}^T)_i)_j$
 8:     $p_{ij}$ computes $(\mathbf{W}_i)_j = \underset{\tilde{\mathbf{W}} \geqslant 0}{\operatorname{argmin}} \left\| \tilde{\mathbf{W}}(\mathbf{HH}^T) - ((\mathbf{AH}^T)_i)_j \right\|$
    /* Compute H given W */
 9:     $p_{ij}$ computes $\mathbf{X}_{ij} = (\mathbf{W}_i)_j{}^T(\mathbf{W}_i)_j$
10:     compute $\mathbf{W}^T\mathbf{W} = \sum_{i,j} \mathbf{X}_{ij}$ using all-reduce across all procs
                                        ▷ $\mathbf{W}^T\mathbf{W}$ is $k \times k$ and symmetric
11:     $p_{ij}$ collects $\mathbf{W}_i$ using all-gather across proc rows
12:     $p_{ij}$ computes $\mathbf{Y}_{ij} = \mathbf{W}_i{}^T\mathbf{A}_{ij}$
                                        ▷ $\mathbf{Y}_{ij}$ is $k \times n/p_c$
13:     compute $(\mathbf{W}^T\mathbf{A})^j = \sum_i \mathbf{Y}_{ij}$ using reduce-scatter across proc columns to achieve column-wise distribution of $(\mathbf{W}^T\mathbf{A})^j$
                                        ▷ $p_{ij}$ owns $k \times n/p$ submatrix $((\mathbf{W}^T\mathbf{A})^j)^i$
14:     $p_{ij}$ computes $(\mathbf{H}^j)^i = \underset{\tilde{\mathbf{H}} \geqslant 0}{\operatorname{argmin}} \left\| (\mathbf{W}^T\mathbf{W})\tilde{\mathbf{H}} - ((\mathbf{W}^T\mathbf{A})^j)^i \right\|$
15: end while
Ensure: $\mathbf{W}, \mathbf{H} \approx \underset{\tilde{\mathbf{W}} \geqslant 0, \tilde{\mathbf{H}} \geqslant 0}{\operatorname{argmin}} \|\mathbf{A} - \tilde{\mathbf{W}}\tilde{\mathbf{H}}\|$
Ensure: $\mathbf{W}$ is an $m \times k$ matrix distributed row-wise across processors, $\mathbf{H}$ is a $k \times n$ matrix distributed column-wise across processors
```

In the rest of this section, we derive the per-iteration computation and communication costs, as well as the local memory requirements. We also argue the communication-optimality of the algorithm in the dense case. Table 2 summarizes the results of this section and compares them to Naive.

Computation Cost

Local matrix computations occur at lines 3, 6, 9, and 12. In the case that **A** is dense, each processor performs

$$\frac{n}{p}k^2 + 2\frac{m}{p_r}\frac{n}{p_c}k + \frac{m}{p}k^2 + 2\frac{m}{p_r}\frac{n}{p_c}k = O\left(\frac{mnk}{p}\right)$$

flops.

In the case that **A** is sparse, processor *(i,j)* performs $(m+n)k^2/p$ flops in computing $\mathbf{U}_{ij}$ and $\mathbf{X}_{ij}$, and $4\mathit{nnz}(A_{ij})k$ flops in computing $\mathbf{V}_{ij}$ and $\mathbf{Y}_{ij}$. Local non-negative least squares problems occur at lines 8 and 14. In each case, the symmetric positive semi-definite matrix is *k*k* and the number of columns/rows of length *k* to be computed are *m/p* and *n/p*, respectively. These costs together require $C_{\text{BPP}}(k, (m+n)/p)$ flops. There are computation costs associated with the all-reduce and reduce-scatter collectives, both those contribute only to lower order terms.

Communication Cost

Communication occurs during six collective operations (lines 4, 5, 7, 10, 11, and 13). We use the cost expressions presented in Section 2for these collectives. The communication cost of the all-reduces (lines 4 and 10) is $\alpha \cdot 4\log p + \beta \cdot 2k^2$;

the cost of the two all-gathers (lines 5 and 11) is

$\alpha \cdot \log p + \beta \cdot ((p_r-1)nk/p + (p_c-1)mk/p)$; and

the cost of the two reduce-scatters (lines 7 and 13) is $\alpha \cdot \log p + \beta \cdot ((p_c-1)mk/p + (p_r-1)nk/p)$.

In the case that $m/p<n$, we choose $p_r = \sqrt{np/m} > 1$ and $p_c = \sqrt{mp/n} > 1$, and these communication costs simplify to

$\alpha \cdot O(\log p) + \beta \cdot O(mk/p_r + nk/p_c + k^2) = \alpha \cdot O(\log p) + \beta \cdot O(\sqrt{mnk^2/p} + k^2)$.

In the case that $m/p \geq n$, we choose $p_c=1$, and the costs simplify to $\alpha \cdot O(\log p) + \beta \cdot O(nk)$.

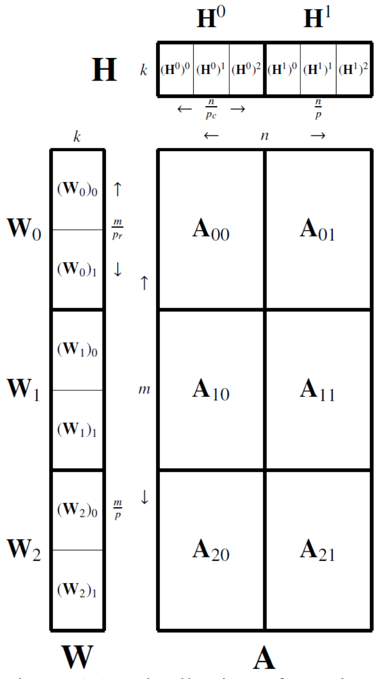


Figure 8.2: Distribution of matrices for HPC-NMF (Algorithm 3), for p_r=3 and p_c = 2. Note that $\mathbf{A}_{ij}$ is $m/p_r * m/p_c$, $\mathbf{W}_i$ is $m/p_r * k$, $(\mathbf{W}_i)_j$ is $m/p * k$, and $\mathbf{H}_j$ is $k * n/p_c$ and $(H_j)_i$ is $k*n/p$.

Memory Requirements

The local memory requirement includes storing each processor's part of matrices **A**, **W**, and **H**.

In the case of dense **A**, this is $mn/p+(m+n)k/p$ words; in the sparse case, processor *(i,j)* requires nnz($\mathbf{A}_{ij}$) words for the input matrix and $(m+n)k/p$ words for the output factor matrices.

Local memory is also required for storing temporary matrices $\mathbf{W}_j$, $\mathbf{H}_i$, $\mathbf{V}_{ij}$, and $\mathbf{Y}_{ij}$, of size $2mk/p_r + 2nk/p_c$ words.

In the dense case, assuming $k<n/p_c$ and $k<m/p_r$, the local memory requirement is no more than a constant times the size of the original data. For the optimal choices of p_r and p_c, this assumption simplifies to $k < \max\left\{\sqrt{mn/p}, m/p\right\}$.

We note that if the temporary memory requirements become prohibitive, the computation of $((\mathbf{AH}^T)_i)_j$ and $((W^TA)_j)_i$ via all-gathers and reduce-scatters can be blocked, decreasing the local memory requirements at the expense of greater latency costs.

While this case is plausible for sparse **A**, we did not encounter local memory issues in our experiments.

Algorithm	Flops	Words	Messages	Memory
Naive	$O\left(\frac{mnk}{p} + (m+n)k^2 + C_{\text{BPP}}\left(\frac{m+n}{p}, k\right)\right)$	$O((m+n)k)$	$O(\log p)$	$O\left(\frac{mn}{p} + (m+n)k\right)$
HPC-NMF ($m/p \geqslant n$)	$O\left(\frac{mnk}{p} + C_{\text{BPP}}\left(\frac{m+n}{p}, k\right)\right)$	$O(nk)$	$O(\log p)$	$O\left(\frac{mn}{p} + \frac{mk}{p} + nk\right)$
HPC-NMF ($m/p < n$)	$O\left(\frac{mnk}{p} + C_{\text{BPP}}\left(\frac{m+n}{p}, k\right)\right)$	$O\left(\sqrt{\frac{mnk^2}{p}}\right)$	$O(\log p)$	$O\left(\frac{mn}{p} + \sqrt{\frac{mnk^2}{p}}\right)$
Lower Bound	–	$\Omega(\min\left\{\sqrt{\frac{mnk^2}{p}}, nk\right\})$	$\Omega(\log p)$	$\frac{mn}{p} + \frac{(m+n)k}{p}$

Table 2: Algorithmic costs for Naive and HPC-NMF assuming data matrix **A** is dense. Note that the communication costs (words and messages) also apply for sparse **A**.

Communication Optimality

In the case that **A** is dense, Algorithm 3 provably minimizes communication costs. Theorem 5.1 establishes the bandwidth cost lower bound for any algorithm that computes W^TA or AH^T each iteration. A latency lower bound of $\Omega(\log p)$ exists in our communication model for any algorithm that aggregates global information [2]. For NMF, this global aggregation is necessary each iteration to compute residual error in the case that **A** is distributed across all *p* processors, for example. Based on the costs derived above, HPC-NMF is communication optimal under the assumption $k < \sqrt{mn/p}$, matching these lower bounds to within constant factors.

Theorem 5.1

Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{W} \in \mathbb{R}^{m\times k}$, *and* $\mathbf{H} \in \mathbb{R}^{k\times n}$ be dense matrices, with $k < n \leqslant m$. If $k < \sqrt{mn/p}$, then any distributed-memory parallel algorithm on *p* processors that load balances the matrix distributions and computes W^TA and/or AH^T must communicate at least $\Omega(\min\{\sqrt{mnk^2/p}, nk\})$ words along its critical path.

Proof

The proof follows directly from [4, Section II.B]. Each matrix multiplication W^TA and AH^T has dimensions $k < n \leqslant m$, so the assumption $k < \sqrt{mn/p}$ ensures that neither multiplication has ``3 large dimensions." Thus, the communication lower bound is either $\Omega(\sqrt{mnk^2/p})$ in the case of $p>m/n$ (or ``2 large dimensions"), or $\Omega(nk)$, in the case of $p<m/n$ (or ``1 large dimension").

If $p<m/n$, then $nk < \sqrt{mnk^2/p}$, so the lower bound can be written as $\Omega(\min\{\sqrt{mnk^2/p}, nk\})$.

We note that the communication costs of Algorithm 3 are the same for dense and sparse data matrices (the data matrix itself is never communicated). In the case that **A** is sparse, this communication lower bound does not necessarily apply, as the required data movement depends on the sparsity pattern of **A**. Thus, we cannot make claims of optimality in the sparse case (for general **A**). The communication lower bounds for W^TA and/or AH^T (where **A** is sparse) can be expressed in terms of hypergraphs that encode the sparsity structure of **A** (Ballard et al. 2015). Indeed, hypergraph partitioners have been used to reduce communication and achieve load balance for a similar problem: computing a low-rank representation of a sparse tensor (without non-negativity constraints on the factors) (Kaya 2015).

8.6 Experiments

In the data mining and machine learning community, there had been a large interest in using Hadoop for large scale implementation. Hadoop does lots of disk I/O and was designed for processing gigantic text files. Many of the real world data sets that is available for research are large scale sparse internet text data such as bag of words, recommender systems, social networks etc. Towards this end, there had been interest towards Hadoop implementation of matrix factorization algorithm (Gemulla et al. 2011; Liao et al. 2014; Liu et al. 2010). However, the use of NMF is beyond the sparse internet data and also applicable for dense real world data such as video, image etc. Hence in order to keep our implementation applicable to wider audience, we chose MPI for distributed implementation. Apart from the application point of view, we decided MPI C++ implementation for other technical advantages that is necessary for scientific application such as (1) it can leverage the recent hardware improvements (2) effective communication between nodes (3) availability of numerically stable BLAS and LAPACK routines etc. We identified a few synthetic and real world datasets to experiment with our MPI implementation and a few baselines to compare our performance.

Experimental Setup

Datasets

We used sparse and dense matrices that are synthetically generated and from real world. We will explain the datasets in this section.

- Dense Synthetic Matrix (*DSYN*): We generate a uniform random matrix of size 172,800*115,200 and add random Gaussian noise. The dimensions of this matrix is chosen such that it is uniformly distributable across processes. Every process will have its own prime seed that is different from other processes to generate the input random matrix **A**.

- Sparse Synthetic Matrix (*SSYN*): We generate a random sparse Erdős–Rényi matrix of the same dimensions 172,800*115,200 as the dense matrix, with density of 0.001. That is, every entry is nonzero with probability 0.001.

- Dense Real World Matrix (*Video*): Generally, NMF is performed in the video data for back ground subtraction to detect the moving objects. The low rank matrix $\hat{\mathbf{A}} \approx \mathbf{W}\mathbf{H}^T$ represents background and the error matrix $\mathbf{A} - \hat{\mathbf{A}}$ has the moving objects. Detecting moving objects has many real-world applications such as traffic estimation, security monitoring, etc. In the case of detecting moving objects, only the last minute or two of video is taken from the live video camera. The algorithm to incrementally adjust the NMF based on the new streaming video is presented in (Kim et al. 2014). To simulate this scenario, we collected a video in a busy intersection of the Georgia Tech campus at 20 frames per second for two minutes. We then reshaped the matrix such that every RGB frame is a column of our matrix, so that the matrix is dense with dimensions 1,013,400 * 2400.

- Sparse Real World Matrix *Webbase* : We identified this dataset of a very large directed sparse graph with nearly 1 million nodes (1,000,005) and 3.1 million edges (3,105,536). The dataset was first reported by Williams et al. (2009). The NMF output of this directed graph will help us understand clusters in graphs. The size of both the real world datasets were adjusted to the nearest dimension for uniformly distributing the matrix.

Machine

We conducted our experiments on ``Edison" at the National Energy Research Scientific Computing Center.

Edison is a Cray XC30 supercomputer with a total of 5,576 compute nodes, where each node has dual-socket 12-core Intel Ivy Bridge processors.

Each of the 24 cores has a clock rate of 2.4 GHz (translating to a peak floating point rate of 460.8 Gflops/node) and private 64KB L1 and 256KB L2 caches; each of the two sockets has a (shared) 30MB L3 cache; each node has 64 GB of memory. Edison uses a Cray ``Aries" interconnect that has a dragonfly topology. Because our experiments use a relatively small number of nodes, we consider the local connectivity: a "blade" comprises 4 nodes and a router, and sets of 16 blades' routers are fully connected via a circuit board backplane (within a ``chassis"). Our experiments do not exceed 64 nodes, so we can assume a very efficient, fully connected network.

Initialization

To ensure fair comparison among algorithms, the same random seed was used across different methods appropriately. That is, the initial random matrix **H** was generated with the same random seed when testing with different algorithms (note that **W** need not be initialized). This ensures that all the algorithms perform the same computations (up to roundoff errors), though the only computation with a running time that is sensitive to matrix values is the local NNLS solve via BPP.

Algorithms

For each of our data sets, we benchmark and compare three algorithms: (1) Algorithm 2 (2) Algorithm 3 with p_r=p and p_c=1 (1D processor grid), and (3) Algorithm 3 with $p_r \approx p_c \approx \sqrt{p}$ (2D processor grid). We choose these three algorithms to confirm the following conclusions from the analysis of Section 5: the performance of a naive parallelization of Naive (Algorithm 2) will be severely limited by communication overheads, and the correct choice of processor grid within Algorithm 3 is necessary to optimize performance. To demonstrate the latter conclusion, we choose the two extreme choices of processor grids and test some data sets where a 1D processor grid is optimal (e.g., the Video matrix) and some where a squarish 2D grid is optimal (e.g., the Webbase matrix).

While we would like to compare against other high-performance NMF algorithms in the literature, the only other distributed-memory implementations of which we're aware are implemented using Hadoop and are designed only for sparse matrices (Liao et al. 2014; Liu et al. 2010; Gemulla et al. 2011; Faloutsos et al. 2014;Yin et al. 2014). We stress that Hadoop is not designed for high performance, requiring disk I/O between steps, so a run time comparison between a Hadoop implementation and a C++/MPI implementation is not a fair comparison of parallel algorithms. To give a qualitative example of differences in run time, the running time of a Hadoop implementation of the MU algorithm on a large sparse matrix of dimension 2^{17} * 2^{16} with $2*10^8$ nonzeros (with k=8) takes on the order of 50 minutes per iteration (Liu et al. 2010); our implementation takes a second per iteration for the synthetic data set (which is an order of magnitude larger in terms of rows, columns, and nonzeros) running on only 24 nodes.

Time Breakdown

To differentiate the computation and communication costs among the algorithms, we present the time breakdown among the various tasks within the algorithms for both performance experiments.

For Algorithm 3, there are three local computation tasks and three communication tasks to compute each of the factor matrices:

- **MM** computing a matrix multiplication with the local data matrix and one of the factor matrices;
- **NLS**, solving the set of NLS problems using BPP;
- **Gram**, computing the local contribution to the Gram matrix;
- **All-Gather**, to compute the global matrix multiplication;
- **Reduce-Scatter**, to compute the global matrix multiplication;
- **All-Reduce**, to compute the global Gram matrix.

In our results, we do not distinguish the costs of these tasks for **W** and **H** separately; we report their sum, though we note that we do not always expect balance between the two contributions for each task. Algorithm 2 performs all of these tasks except the Reduce-Scatter and the All-Reduce; all of its communication is in the All-Gathers.

Algorithmic Comparison

Our first set of experiments is designed primarily to compare the three parallel implementations. For each data set, we fix the number of processors to be 600 and vary the rank *k* of the desired factorization. Because most of the computation (except for NLS) and bandwidth costs are linear in *k* (except for the All-Reduce), we expect linear performance curves for each algorithm individually.

The left side of Figure 8.3 shows the results of this experiment for all four data sets. The first conclusion we draw is that HPC-NMF with a 2D processor grid performs significantly better than the Naive; the largest speedup is 4.4×, for the sparse synthetic data and *k*=10 (a particularly communication bound problem). Also, as predicted, the 2D processor grid outperforms the 1D processor grid on the squarish matrices. While we expect the 1D processor grid to outperform the 2D grid for the tall-and-skinny Video matrix, their performance is comparable; this is because both algorithms are computation bound, as we see from Figure 8.3g, so the difference in communication is negligible.

The second conclusion we can draw is that the scaling with *k* tends to be close to linear, with an exception in the case of the Webbase matrix. We see from Figure 8.3e that this problem spends much of its time in NLS, which does not scale linearly with *k*.

We can also compare HPC-NMF with a 1D processor grid with Naive for squarish matrices (SSYN, DSYN, and Webbase). Our analysis does not predict a significant difference in communication costs of these two approaches (when $m \approx n$), and we see from the data that Naive outperforms HPC-NMF for two of the three matrices (but the opposite is true for DSYN). The main differences appear in the All-Gather versus Reduce-Scatter communication costs and the local MM (all of which are involved in the W^TA computation). In all three cases, our proposed 2D processor grid (with optimal choice of $m/p_r \approx n/p_c$) performs better than both alternatives.

Strong Scalability

The goal of our second set of experiments is to demonstrate the (strong) scalability of each of the algorithms. For each data set, we fix the rank *k* to be 50 and vary the number of processors (this is a strong-scaling experiment because the size of the data set is fixed). We run our experiments on {24,96,216,384,600} processors/cores, which translates to {1,4,9,16,25} nodes. The dense matrices are too large for 1 or 4 nodes, so we benchmark only on {216,384,600} cores in those cases.

The right side of Figure 8.3 shows the scaling results for all four data sets, and Table 8.3 gives the overall per-iteration time for each algorithm, number of processors, and data set. We first consider the HPC-NMF algorithm with a 2D processor grid: comparing the performance results on 25 nodes (300 cores) to the 1 node (24 cores), we see nearly perfect parallel speedups. The parallel speedups are 23× for SSYN and 28× for the Webbase matrix. We believe the superlinear speedup of the Webbase matrix is a result of the running time being dominated by NLS; with more processors, the local NLS problem is smaller and more likely to fit in smaller levels of cache, providing better performance. For the dense matrices, the speedup of HPC-NMF on 25 nodes over 9 nodes is 2.7× for DSYN and 2.8× for Video, which are also nearly linear.

In the case of the Naive algorithm, we do see parallel speedups, but they are not linear. For the sparse data, we see parallel speedups of 10× and 11× with a 25× increase in number of processors. For the dense data, we observe speedups of 1.6× and 1.8× with a 2.8× increase in the number of processors. The main reason for not achieving perfect scaling is the unnecessary communication overheads.

8.7 Conclusion

In this research, we propose a high-performance distributed-memory parallel algorithm that computes an NMF factorization by iteratively solving alternating non-negative least squares (NLS) subproblems.

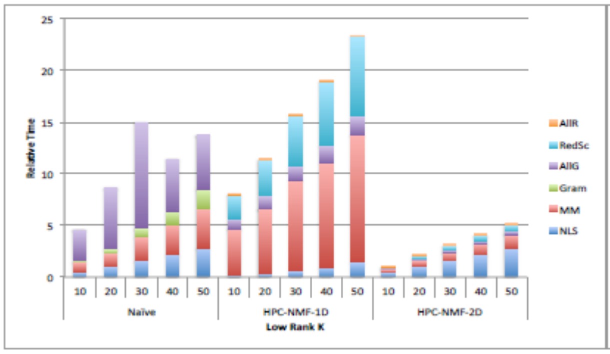
We show that by carefully designing a parallel algorithm, we can avoid communication overheads and scale well to modest numbers of cores.

For the datasets on which we experimented, we showed that an efficient implementation of a naive parallel algorithm spends much of its time in interprocessor communication. In the case of HPC-NMF, the problems remain computation bound on up to 600 processors, typically spending most of the time in local NLS solves. We focus in this work on BPP, which is more expensive per-iteration than alternative methods like MU and HALS, because it has been shown to reduce overall running time in the sequential case by requiring fewer iterations (Kim and Park 2011). Because most of the time per iteration of HPC-NMF is spent on local NLS, we believe further empirical exploration is necessary to confirm the advantages of BPP in the parallel case. We note that if we use MU or HALS for local NLS, the relative cost of interprocessor communication will grow, making the communication efficiency of our algorithm more important.

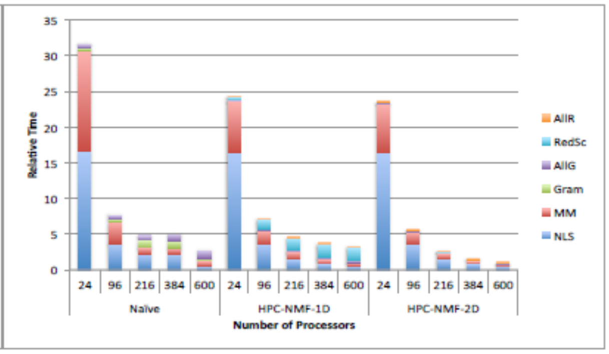
In future work, we would like to extend this algorithm to dense and sparse tensors, computing the CANDECOMP/PARAFAC decomposition in parallel with non-negativity constraints on the factor matrices. We would also like to explore more intelligent distributions of sparse matrices: while our 2D distribution is based on evenly dividing rows and columns, it does not necessarily load balance the nonzeros of the matrix, which can lead to load imbalance in MM. We are interested in using graph and hypergraph partitioning techniques to load balance the memory and computation while at the same time reducing communication costs as much as possible. Finally, we have not yet reached the limits of the scalability of HPC-NMF; we would like to expand our benchmarks to larger numbers of nodes on the same size datasets to study performance behavior when communication costs completely dominate the running time.

	Naive				HPC-NMF-1D				HPC-NMF-2D			
Cores	**DSYN**	**SSYN**	**Video**	**Webbase**	**DSYN**	**SSYN**	**Video**	**Webbase**	**DSYN**	**SSYN**	**Video**	**Webbase**
24		6.5632		48.0256		5.0821		52.8549		4.8427		84.6286
96		1.5929		18.5507		1.4836		14.5873		1.1147		16.6966
216	2.1819	0.6027	2.7899	7.1274	2.1548	0.9488	4.7928	9.2730	1.5283	0.4816	1.6106	7.4799
384	1.2594	0.6466	2.2106	5.1431	1.2559	0.7695	3.8295	6.4740	0.8620	0.2661	0.8963	4.0630
600	1.1745	0.5592	1.7583	4.6825	0.9685	0.6666	0.5994	6.2751	0.5519	0.1683	0.5699	2.7376

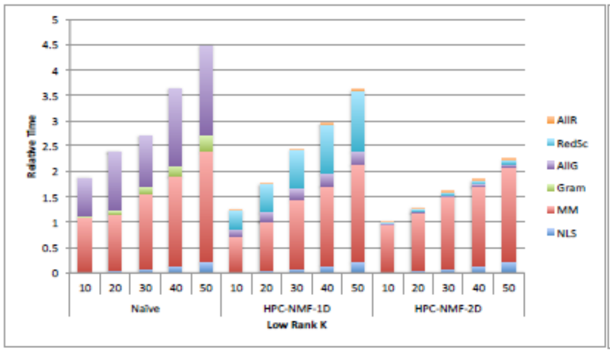
Table 8.3: Per-iteration running times of parallel NMF algorithms for k=50.



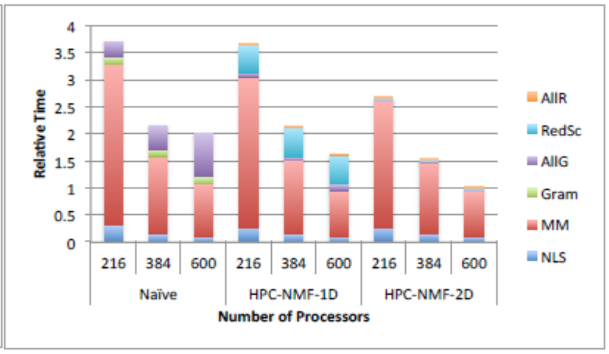
(a) Sparse Synthetic (SSYN) Comparison



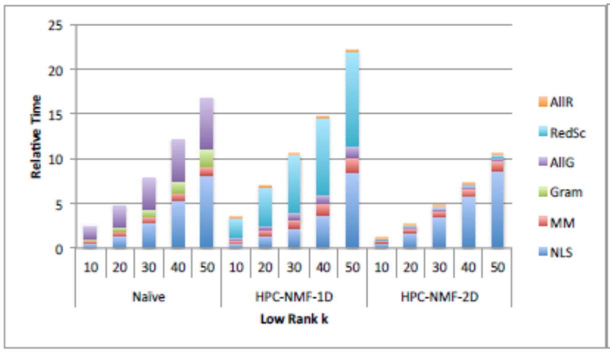
(b) Sparse Synthetic (SSYN) Scaling



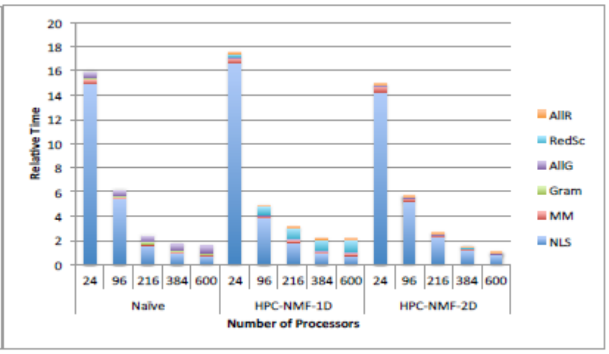
(c) Dense Synthetic (DSYN) Comparison



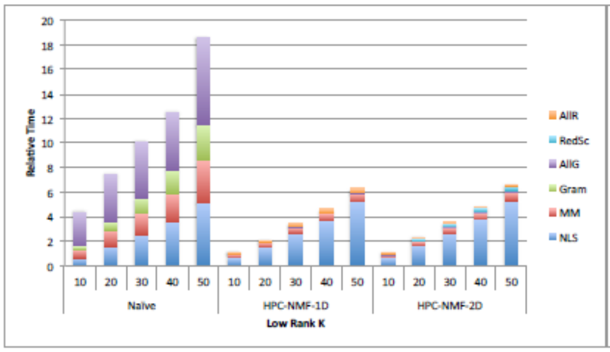
(d) Dense Synthetic (DSYN) Scaling



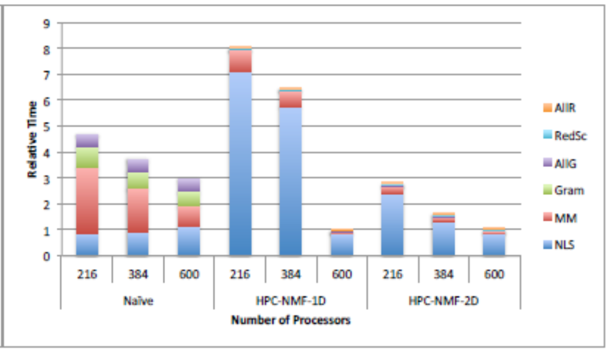
(e) Webbase Comparison



(f) Webbase Scaling



(g) Video Comparison



(h) Video Scaling

Figure 8.3: Experiments on Sparse and Dense Datasets

9 References

Allen, G. 2007. “Building a Dynamic Data Driven Application System for Hurricane Forecasting.” In Computational Science – ICCS 2007, Edited by Y. Shi, G. D. v. Albada, J. Dongarra, and P. M. A. Sloot, 1034–1041. Berlin: Springer Berlin Heidelberg.

Aschenbruck, N., et al. 2010. Bonnmotion: a mobility scenario generation and analysis tool. Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Ayani, R. 1988. A Parallel Simulation Scheme Based on Distances Between Objects. Royal Institute of Technology, Department of Telecommunication Systems-Computer Systems.

Ballard, G., A. Druinsky, N. Knight, and O. Schwartz. Brief announcement: Hypergraph partitioning for parallel sparse matrix-matrix multiplication. In Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA ’15, pages 86–88, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3588-1. . URL http://doi.acm.org/10.1145/2755573.2755613.

Bhatti, K., C. Belleudy, and M. Auguin, "Power management in real time embedded systems through online and adaptive interplay of DPM and DVFS policies," presented at the International Conference on Embedded and Ubiquitous Computing, Hong Kong, China, 2010.

Brun, C., T. Artés, T. Margalef, and A. Cortés, "Coupling wind dynamics into a DDDAS forest fire propagation prediction system," in *Proceedings of the International Conference on Compuational Science*, 2012.

Bryant, R. E., "Simulation of Packet Communication Architecture Computer Systems," M.S. thesis, MIT-LCS-TR-188, Computer Science Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1977.

Cai, W., F. Lee, and L. Chen. 1999. "An auto-adaptive dead reckoning algorithm for distributed interactive simulation." In Proceedings of the thirteenth workshop on Parallel and distributed simulation. 82-89. IEEE Computer Society.

Chan, E., M. Heimlich, A. Purkayastha, and R. van de Geijn. Collective communication: theory, practice, and experience. Concurrency and Computation: Practice and Experience, 19(13):1749–1783, 2007. ISSN 1532-0634. . URL http://dx.doi.org/10.1002/cpe.1206.

Chan, Y.-M., S.-S. Huang, L.-C. Fu, and P.-Y. Hsiao, "Vehicle Detection Under Various Lighting Conditions by Incorporating Particle Filter," in *Intelligent Transportation Systems Conference*, 2007, pp. 534-539.

Chandy, K., & Misra, J. (1979). Distributed simulation: A case study in design and verification of distributed programs. IEEE Transactions on Software Engineering.

Chaturvedi, A., A. Mellema, S. Filatyev, and J. Gore. 2006. “DDDAS for Fire and Agent Evacuation Modeling of the Rhode Island Nightclub Fire.” In Computational Science – ICCS 2006, Edited by V. N. Alexandrov, G. D. v. Albada, P. M. A. Sloot, and J. Dongarra, 433–439. Berlin: Springer Berlin Heidelberg.

Cheng, C., R. Jain, E. Van Den Berg. 2003. "Location Prediction Algorithms for Mobile Wireless Systems." Wireless Internet Handbook: Technologies, Standards, and Applications.

Cho, K.-M., C.-H. Liang, J.-Y. Huang, and C.-S. Yang, "Design and implementation of a general purpose power-saving scheduling algorithm for embedded systems," presented at the IEEE International Conference on Signal Processing, Communications and Computing, Xi'an, China, 2011.

Cichocki, A., R. Zdunek, A. H. Phan, and S.-i. Amari. Nonnegative matrix and tensor factorizations: applications to exploratory multiway data analysis and blind source separation. Wiley, 2009.

Çinlar, E. 1975. Introduction to Stochastic Processes. Englewood Cliffs, NJ: Printice-Hall.

Cortial, J., C. Farhat, L. J. Guibas, and M. Rajashekhar. 2007. “Compressed Sensing and Time-Parallel Reduced-Order Modeling for Structural Health Monitoring Using a DDDAS.” In Computational Science – ICCS 2007, Edited by Y. Shi, G. D. v. Albada, J. Dongarra, and P. M. A. Sloot, 1171–1179. Berlin: Springer Berlin Heidelberg.

Czechowski K., and R. Vuduc, "A Theoretical Framework for Algorithm-Architecture Co-design," presented at the Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, 2013.

Darema, F. 2004. “Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements.” In Computational Science – ICCS 2004, Edited by M. Bubak, G. D. v. Albada, P. M. A. Sloot, and J. Dongarra, 662–669. Berlin: Springer Berlin Heidelberg.

Davis, W. J. 1998. “On-Line Simulation: Need and Evolving Research Requirements.” In Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice, Edited by J. Banks, 465–518. New York: Wiley.

Demmel, J., D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger. Communication-optimal parallel recursive rectangular matrix multiplication. In Proceedings of the 27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS '13, pages 261–272, 2013.

Dongarra, J., H. Ltaief, P. Luszczek, and V. M. Weaver, "Energy Footprint of Advanced Dense Numerical Linear Algebra using Tile Algorithms on Multicore Architecture," presented at the The 2nd International Conference on Cloud and Green Computing, 2012.

Dowling, R., J. Holland, and A. Huang. Guidelines for Applying Traffic Microsimulation Modeling Software. California Department of Transportation. 2002.

Dowling, R., A. Skabardonis, and V. Alexiadis. Traffic Analysis Toolbox: Guidelines for Applying Traffic Microsimulation Modeling Software. Federal Highway Administration. 2004. [Available from http://ops.fhwa.dot.gov/trafficanalysistools/tat_vol3/Vol3_Guidelines.pdf].

Dowling, R., A. Skabardonis, J. Halkias, G. McHale, and G. Zammit. "Guidelines for Calibration of Microsimulation Models: Framework and Applications." Transportation Research Record 1876. 2004.

Douglas, C. C., R. A. Lodder, J. D. Beezley, J. Mandel, R. E. Ewing, Y. Efendiev, G. Qin, M. Iskandarani, J. Coen, A. Vodacek, M. Kritz, and G. Haase. 2006. "DDDAS Approaches to Wildland Fire Modeling and Contaminant Tracking." In Proceedings of the 2006 Winter Simulation Conference, 2117–2124.

Eeckhout, J.. 2004. "Gibrat's Law for (All) Cities." The American Economic Review, Vol. 94, No. 5.

Esmaeilzadeh, H., T. Cao, X. Yang, S. M. Blackburn, and K. S. McKinley, "Looking back and looking forward: power, performance, and upheaval," Commun. ACM, vol. 55, pp. 105--114, July 2012.

Fairbanks, J. P., R. Kannan, H. Park, and D. A. Bader. Behavioral clusters in dynamic graphs. Parallel Computing, 47:38–50, 2015.

Faloutsos, C., A. Beutel, E. P. Xing, E. E. Papalexakis, A. Kumar, and P. P. Talukdar. Flexi-fact: Scalable flexible factorization of coupled tensors on hadoop. In Proceedings of the SDM, pages 109–117, 2014. URL http://epubs.siam.org/doi/abs/10.1137/1. 9781611973440.13.

Farhat, C., J. G. Michopoulos, F. K. Chang, L. J. Guibas, and A. J. Lew. 2006. "Towards a Dynamic Data Driven System for Structural and Material Health Monitoring." In Computational Science – ICCS 2006, Edited by V. N. Alexandrov, G. D. v. Albada, P. M. A. Sloot, and J. Dongarra, 456–464. Berlin: Springer Berlin Heidelberg.

Feng, X., R. Ge, and K. W. Cameron, "Power and Energy Profiling of Scientific Applications on Distributed Systems," presented at the Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2005.

Ferenci, S., Fujimoto, R., Ammar, M., Perumalla, K., & Riley, G. 2002. Updateable simulation of communication networks. Proceedings of the sixteenth workshop on Parallel and distributed simulation.

FHWA. Next Generation Simulation. 2007. [Available from http://ngsim-community.org/].

Freeh, V. W., D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, et al., "Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications," IEEE Trans. Parallel Distrib. Syst., vol. 18, pp. 835--848, June 2007.

Fujimoto, R., A. Guin, M. Hunter, H. Park, R. Kannan, G. Kanitkar, M. Milholen, S. Neal, P. Pecher. 2014. "A Dynamic Data Driven Application System for Vehicle Tracking." International Conference on Computational Science, Dynamic Data Driven Application Systems Workshop, June 2014.

Fujimoto, R., D. Lunceford, E. Page, and A. M. Uhrmacher. 2002. "Grand Challenges for Modeling and Simulation." Dagstuhl Seminar Report 350, Schloss Dagstuhl, Dagstuhl, Germany.

Fujimoto, R., M. Hunter, J. Sirichoke, M. Palekar, H. Kim, and W. Suh. 2007. "Ad Hoc Distributed Simulations." In Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, 15–24.

Ge, R., X. Feng, and K. W. Cameron, "Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters," presented at the Proceedings of the 2005 ACM/IEEE conference on Supercomputing, Washington, DC, USA, 2005.

Ge, R., X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," IEEE Transactions on Parallel and Distributed Systems (TPDS), vol. 21, pp. 658-671, May 2010.

Gemulla, R., E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In Proceedings of the KDD, pages 69–77. ACM, 2011.

Gonzalez R., and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," IEEE Journal of Solid-State Circuits, vol. 31, pp. 1277-1284, September 1996.

Gordon, Mark, Lide Zhang, Birjodh Tiwana, Robert Dick, Zhuoqing Morley Mao, and Lei Yang. "PowerTutor." *PowerTutor*. N.p., 2009. Web. 20 Nov. 2013.

Grasso, I., P. Radojkovic, N. Rajovic, I. Gelado, and A. Ramirez, "Energy Efficient HPC on Embedded SoCs: Optimization Techniques for Mali GPU," presented at the Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, 2014.

Hellinga, B. Requirements for the Calibration of Traffic Simulation Models. In Canadian Society for Civil Engineering 1998 Annual Conference, 1998.

Henclewood, D. Real-Time Estimation of Arterial Performance Measures Using a Data-Driven Microscopic Traffic Simulation Technique. Ph.D. Thesis, Georgia Institute of Technology, 2012.

Hoeller, A., L. Wanner, and A. Fröhlich, "A hierarchical approach for power management on mobile embedded systems," in From Model-Driven Design to Resource Management for Distributed Embedded Systems, ed, 2006, pp. 265–274.

Hollander, Y., and R. Liu. "The Principles of Calibrating Traffic Microsimulation models." Transportation 35 (3) 347-362. 2008.

Holmes, V. 1978. Parallel algorithms for multiple processor architectures. Ph.D. dissertation, Comp. Science Dept.. Univ. Texas at Austin.

Hoyer, P. O.. Non-negative matrix factorization with sparseness constraints.mJMLR, 5:1457–1469, 2004.

Hua S., and G. Qu, "Approaching the Maximum Energy Saving on Embedded Systems with Multiple Voltages," presented at the IEEE/ACM International Conference on Computer-Aided Design, 2003.

Huang, Y.-L., C. Alexopoulos, M. Hunter, and R. Fujimoto. 2012. "Ad Hoc Distributed Simulation Methodology for Open Queueing Networks." Simulation 88(7): 784–800.

Hunter, M., H. K. Kim, W. Suh, R. Fujimoto, J. Sirichoke, and M. Palekar. 2009. "Ad Hoc Distributed Dynamic Data-Driven Simulations of Surface Transportation Systems." Simulation 85(4): 243–255.

Hunter, M., J. Sirichoke, R. Fujimoto, and Y.-L. Huang. 2009. "Embedded Ad Hoc Distributed Simulation for Transportation System Monitoring and Control." In Proceedings of the 2009 INFORMS Simulation Society Research Workshop, 15–19.

Hybinette, M., & Fujimoto, R. M. 2001. Cloning parallel simulations. ACM Transactions on Modeling and Computer Simulation.

IEEE Std 1516.1-2010, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) -- Interface Specification," ed. New York, NY: Institute of Electrical and Electronics Engineers, Inc., 2010.

Kamrani, F. and R. Ayani. 2007. "Using On-Line Simulation for Adaptive Path Planning of UAVs." In Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications, 167–174.

Kaya O., and B. Uc¸ar. Scalable sparse tensor decompositions inndistributed memory systems. Technical Report RR-8722, INRIA, May 2015.

Keckler, S. W., W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," Micro, IEEE, vol. 31, pp. 7 -17, sept.-oct. 2011.

Kiesling, T., and J. Luthi. 2005. "Towards Time-Parallel Road Traffic Simulation." In Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation (PADS '05). IEEE Computer Society, Washington, DC, USA, 7-15.

Kim, H., and H. Park. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. Bioinformatics, 23(12):1495–1502, 2007.

Kim, J., and H. Park. Fast nonnegative matrix factorization: An activeset- like method and comparisons. SIAM Journal on Scientific Computing, 33(6):3261–3281, 2011.

Kim, J., Y. He, and H. Park. Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. Journal of Global Optimization, 58(2):285–319, 2014.

Knight, D., Q. Ma, T. Rossman, and Y. Jaluria, "Evaluation of Fluid-thermal Systems by Dynamic Data Driven Application Systems – Part II," in *Proceedings of the 2007 International Conference on Computational Science*, 2007, pp. 1189–1196.

Knyazev, R. Kremens, V. Kulkarni, G. Qin, A. Vodacek, J. Wu, W. Zhao, and A. Zornes. 2005. "Towards a Dynamic Data Driven Application System for Wildfire Simulation." In Computational Science – ICCS 2005, Edited by V. S. Sunderam, G. D. v. Albada, P. M. A. Sloot, and J. J. Dongarra, 632–639. Berlin: Springer Berlin Heidelberg.

Krumm, J., "Real Time Destination Prediction Based On Efficient Routes," in *SAE International*, 2006.

Kuang, D., C. Ding, and H. Park. Symmetric nonnegative matrix factorization for graph clustering. In Proceedings of SDM, pages 106–117, 2012.

Kvam, P., and B. Vidakovic. Nonparametric Statistics with Applications to Science and Engineering. 2007.

Laasonen, K., "Route Prediction from Cellular Data," in *Workshop on Context-Awareness for Proactive Systems*, 2005.

Lee, E. A. 2008. “Cyber Physical Systems: Design Challenges.” In Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing, 363–369.

Lee, K., et al. (2009). Slaw: A new mobility model for human walks. INFOCOM 2009, IEEE, IEEE.

Lewis, P. A. W., and G. S. Shedler. 1979. “Simulation of Nonhomogeneous Poisson Processes by Thinning.” Naval Research Logistics Quarterly 26(3): 403–413.

Liao, R., Y. Zhang, J. Guan, and S. Zhou. Cloudnmf: A mapreduce implementation of nonnegative matrix factorization for large-scale biological datasets. Genomics, proteomics & bioinformatics, 12(1): 48–51, 2014.

Liu, C., H.-c. Yang, J. Fan, L.-W. He, and Y.-M. Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In Proceedings of the WWW, pages 681–690. ACM, 2010.

Low, M. Y. H., K. W. Lye, P. Lendermann, S. J. Turner, R. T. W. Chim, and S. H. Leo. 2005. “An Agent-Based Approach for Managing Symbiotic Simulation of Semiconductor Assembly and Test Operation.” In Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems, 85–92.

Lubachevsky, B. D., "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks," Communications of the ACM, vol. 32, pp. 111-123, 1989.

Madey, G. R., M. B. Blake, C. Poellabauer, H. Lu, R. R. McCune, and Y. Wei, "Applying DDDAS Principles to Command, Control and Mission Planning for UAV Swarms," in *Proceedings of the International Conference on Compuational Science*, 2012.

Madey, G. R., G. Szabo, and A.-L. Barabási, "WIPER: The Integrated Wireless Phone Based Emergency Response System," in *Proceedings of the 2006 International Conference on Computational Science*, 2006, pp. 417–424.

Madey, G. R., A.-L. Barabási, N. V. Chawla, M. Gonzalez, D. Hachen, B. Lantz*, et al.*, "Enhanced Situational Awareness: Application of DDDAS Concepts to Emergency and Disaster Management," in *Proceedings of the 2007 International Conference on Computational Science*, 2007, pp. 1090–1097.

Mandel, J., J. D. Beezley, A. K. Kochanski, V. Y. Kondratenko, and M. Kim, "Assimilation of Perimeter Data and Coupling with Fuel Moisture in a Wildland Fire – Atmosphere DDDAS," in *Proceedings of the International Conference on Compuational Science*, 2012.

Mandel, J., L. S. Bennethum, M. Chen, J. L. Coen, C. C. Douglas, L. P. Franca, C. J. Johns, M. Kim, A. V.

Marin, M., V. Gil-Costa, C. Bonacic, and R. Solar. 2013. "Approximate Parallel Simulation of Web Search Engines." In PADS. 189–200.

Mikluščák, T., Gregor, M., & Janota, A. 2012. Using Neural Networks for Route and Destination Prediction in Intelligent Transport Systems. Telematics in the Transport Environment .

Miller, D. Developing a Procedure to Identify Parameters for Calibration of a VISSIM Model. Master Thesis, Georgia Institute of Technology. 2009.

Miller, D., D. Henclewood, M. Rodgers, and M. Hunter. Parameter Selection Procedure for Signalized Arterial Simulation Calibration. In 91st TRB Annual Meeting. 2012.

Mooney, C., and R. Duval. Bootstrapping: A Nonparametric Approach to Statistical Inference. 1993.

Munjal, A., et al. (2011). "SMOOTH: a simple way to model human walks." ACM SIGMOBILE Mobile Computing and Communications Review **14**(4): 34-36.

Nagel, Kai, and M. Schreckenberg. 1992. "A cellular automaton model for freeway traffic." Journal de physique I 2. No. 12 . 2221-2229.

Neal, S., G. Kanitkar, and R. M. Fujimoto, "Power Consumption of Data Distribution Management for On-Line Simulations," presented at the Principles of Advanced Discrete Simulation, Denver, Co., 2014.

NGSIM Community. 2014. NGSIM Community Home. Accessed May 13th. http://ngsim-community.org.

Nicol, D. M., "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations," Journal of the Association for Computing Machinery, vol. 40, pp. 304-333, June 1993.

Nicol, D. M., "Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks," SIGPLAN Notices, vol. 23, pp. 124-137, 1988.

Niu, L., and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real- time systems," presented at the international conference on Compilers, architecture, and synthesis for embedded systems, 2004.

NS-3, "The ns3 network simulator," http://www.nsnam.org/, 2011.

Oketch, T, and M. Carrick. Calibration and validation of a micro-simulation model in network analysis. In 84th TRB Annual Meeting, 2005.

Pan, K., et al. 2009. Implementation of data distribution management services in a service oriented HLA RTI. Winter Simulation Conference, Winter Simulation Conference.

Park, B., and J. Won. Microscopic Simulation Model Calibration and Validation Handbook. Virginia DOT. 2006. [Available from http://www.virginiadot.org/vtrc/main/online_reports/pdf/07-cr6.pdf].

Park, B., and J. Schneeberger. “Calibration and Validation Case Study of VISSIM Simulation Model for a Coordinated Actuated Signal System.” Transportation Research Record 1856. pp. 185-192. 2003.

Park, B., J. Won, and I. Yun. “Application of Microscopic Simulation Model Calibration and Validation Procedure: Case Study of Coordinated Actuated Signal System.” Transportation Research Record 1978. 2006.

Patra, A. K., M. Bursik, J. Dehn, M. Jones, M. Pavolonis, E. B. Pitman*, et al.*, "A DDDAS Framework for Volcanic Ash Propagation and Hazard Analysis," in *Proceedings of the International Conference on Compuational Science*, 2012.

Pauca, V. P., F. Shahnaz, M. W. Berry, and R. J. Plemmons. Text mining using nonnegative matrix factorizations. In Proceedings of SDM, 2004.

Peacock, J., Wang, J., & Manning, E. (1978). Distributed simulation using a network of processors. Proceedings of the 3rd Berkeley Workshop on Distributed Data Management and Computer Networks.

Pecher, P. K., Hunter, M., & Fujimoto, R. M. (2014). Past and Future Trees: Structures for Predicting Vehicle Trajectories in Real-Time. 2014 Winter Simulation Conference.

Pournajaf, L., Xiong, L., & Sunderam, V. (2014). Dynamic Data Driven Crowd Sensing Task Assignment. ICCS 2014. 14th International Conference on Computational Science.

PTV. VISSIM 5.10 User Manual. 2011.

Quan G., and X. Hu, "Energy efficient fixed- priority scheduling for real-time systems on variable voltage processors," presented at the Design Automation Conference, 2001.

Rajovic, N., P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?," presented at the Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, New York, NY, USA, 2013.

Rajovic, N., A. Rico, J. Vipond, I. Gelado, N. Puzovic, and A. Ramirez, "Experiences with mobile processors for energy efficient HPC," presented at the Design, Automation Test in Europe Conference Exhibition (DATE), 2013, 2013.

Rak, S. J. and D. J. Van Hook 1996. Evaluation of grid-based relevance filtering for multicast group assignment. Proc. of 14th DIS workshop, Citeseer.

Riley, G. F. and T. R. Henderson 2010. The ns-3 network simulator. Modeling and Tools for Network Simulation, Springer**:** 15-34.

Roess, R., W. McShane, and E. Prassas, Traffic Engineering. 2003.

Saewong S., and R. Rajkumar, "Practical voltage- scaling for fixed-priority rt-systems," presented at the IEEE Real- Time and Embedded Technology and Applications Symposium, 2003.

Seung, D., and L. Lee. Algorithms for non-negative matrix factorization. NIPS, 13:556–562, 2001.

Schrank, D., T. Lomax, and B. Eisele. Urban Mobility Report 2011. Texas Transportation Institute.

Shi, W., K. S. Perumalla, and R. M. Fujimoto, "Power-aware State Dissemination in Mobile Distributed Virtual Environments," in Workshop on Parallel and Distributed Simulation, San Diego, 2003.

Sinclair, A. J., R. J. Prazenica, and D. E. Jeffcoat. 2008. "Optimal and Feedback Path Planning for Cooperative Attack." Journal of Guidance, Control, and Dynamics, Vol. 31, No. 6. 1708-1715.

Sivaraman, S., and M. M. Trivedi, "A General Active-Learning Framework for On-Road Vehicle Recognition and Tracking," *IEEE Transactions on Intelligent Transportation Systems,* vol. 11, pp. 267-276, 2010.

Stanisic, L., B. Videau, J. Cronsioe, A. Degomme, V. Marangozova-Martin, A. Legrand, et al., "Performance analysis of HPC applications on low-power embedded platforms," presented at the Design, Automation Test in Europe Conference Exhibition (DATE), 2013, 2013.

Suh, W., M. Hunter, and R. M. Fujimoto, "Ad Hoc Distributed Simulation for Transportation System Monitoring and Near-Term Prediction," *Simulation Modeling Practice and Theory,* vol. 41, pp. 1-14, 2014.

Tan, G., et al. 2001. An agent-based DDM for high level architecture. Proceedings of the fifteenth workshop on Parallel and distributed simulation, IEEE Computer Society.

Tan, G., et al. 2000. A hybrid approach to data distribution management. Distributed Simulation and Real-Time Applications, 2000.(DS-RT 2000). Proceedings. Fourth IEEE International Workshop on, IEEE.

Tang, J. C., et al. 2011. "Reflecting on the DARPA red balloon challenge." Communications of the ACM **54**(4): 78-85. [13] Wood, D. D. (2002). Implementation of DDM in the MAK High Performance RTI. Proceedings of the Simulation Interoperability Workshop, Citeseer.

Thakur, R., R. Rabenseifner, andW. Gropp. Optimization of collective communication operations in MPICH. International Journal of High Performance Computing Applications, 19(1):49–66, 2005. . URL http://hpc.sagepub.com/content/19/1/49.abstract.

Toledo, T., M. Ben-Akiva, D. Darda, M. Jha, and H. Koutsopoulos. “Calibration of microscopic traffic simulation models with aggregate data.” Transportation Research Record 1876. 2004.

Unsal, O. S., "System-Level Power-Aware Computing In Complex Real-Time and Multimedia Systems," Doctor of Philosophy Doctoral Dissertation, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, 2008.

Viola, P. and M. J. Jones, "Robust real-time face detection," *International journal of computer vision,* vol. 57, pp. 137-154, 2004.

Voronkov, A., and F. Darema. 2004. "Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements." In International Conference on Computational Science. 662-669.

Wang, Y.-X., and Y.-J. Zhang. Nonnegative matrix factorization: A comprehensive review. TKDE, 25(6):1336–1353, June 2013. ISSN 1041-4347.

Williams, S., L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix?vector multiplication on emerging multicore platforms. Parallel Computing, 35(3):178 – 194, 2009.

Williams, S., A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," Commun. ACM, vol. 52, pp. 65--76, April 2009.

D. Wood, "Implementation of DDM in the MAK High Performance RTI," *Simulation Interoperability Workshop*, 2002.

Xue, A. Y., Zhang, R., Zheng, Y., Xie, X., Huang, J., & Xu, Z. (2013). Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. Data Engineering (ICDE), 2013 IEEE 29th International Conference.

Ye, T., H. T. Kaur, S. Kalyanaraman, and M. Yuksel. 2008. "Large-Scale Network Parameter Con?guration Using an On-Line Simulation Framework." IEEE/ACM Transactions on Networking 16(4): 777–790.

Yin, J., L. Gao, and Z. Zhang. Scalable nonnegative matrix factorization with block-wise updates. In Machine Learning and Knowledge Discovery in Databases, volume 8726 of LNCS, pages 337–352, 2014. ISBN 978-3-662-44844-1.

Zhang, M., J. Ma, and H. Dong. Developing Calibration Tools for Microscopic Traffic Simulation Final Report: Calibration Framework and Calibration of Local/Global Driving Behavior and Departure/Route Choice Model Parameters. California PATH Research Report. 2008.

Zhang, M., and J. Ma. Developing Calibration Tools for Microscopic Traffic Simulation Final Report: Overview Methods and Guidelines on Project Scoping and Data Collection. California PATH Research Report. 2008. Chu, L., H. Liu, J. Oh, and W. Recker. A Calibration Procedure for Microscopic Traffic Simulation. In 83rd TRB Annual Meeting, 2004.

1.

1. Report Type

Final Report

Primary Contact E-mail
Contact email if there is a problem with the report.

fujimoto@cc.gatech.edu

Primary Contact Phone Number
Contact phone number if there is a problem with the report

404-894-5615

Organization / Institution name

Georgia Institute of Technology

Grant/Contract Title
The full title of the funded effort.

Dynamic Systems for Individual Tracking via Heterogeneous Information Integration and Crowd Source Distributed Simulation

Grant/Contract Number
AFOSR assigned control number. It must begin with "FA9550" or "F49620" or "FA2386".

FA9550-13-1-0100

Principal Investigator Name
The full name of the principal investigator on the grant or contract.

Richard Fujimoto

Program Manager
The AFOSR Program Manager currently assigned to the award

Frederica Darema

Reporting Period Start Date

03/14/2013

Reporting Period End Date

08/31/2015

Abstract

Tracking the movement of individuals in complex urban environments using mobile sensors is a challenging, but important problem in applications such as law enforcement, homeland security and defense. The Dynamic Data Driven Application Systems (DDDAS) paradigm offers a natural approach to attacking this problem. This two-year research project explored new computational technologies based on the DDDAS paradigm that could be applied to track vehicles in real time. Several technical challenges were addressed in areas such as dynamic data-driven distributed simulation, analytics to exploit crowd-sourced and online sensor data, vehicle trajectory analysis and prediction, and image analysis for recognition of vehicles. New avenues of research in distributed simulation focusing on energy consumption issues were explored as part of this research in order to create effective distributed simulations executing on mobile computing platforms, an area not widely studied to date in the distributed simulation research community.

A computational architecture was developed based on the DDDAS approach to frame and address computational issues in the vehicle tracking problem. This architecture includes image processing algorithms for vehicle detection, predictive data analytics coupled with distributed simulations to assess

likely future locations of the vehicle, and reconfiguration of the sensor network to maintain tracking.

This research project achieved several advances to further the development and exploitation of DDDAS technologies. Principal research accomplishments and results include:

• Transient response of data-driven distributed simulation. We envision the use of distributed simulations operating on mobile devices in close proximity to the physical system, e.g., simulations executing within a mobile sensor network that may be used to automatically reconfigure the sensor network. Methods to improve predictions of transient behaviors by data-driven distributed simulations were developed and analyzed. An important, distinguishing aspect of this research concerns the use of a distributed simulation approach that emphasizes realization within the sensor network itself to guide online decision making rather than reliance on centralized back-end computing facilities.

• Methods for on-line data driven calibration of traffic simulation. An approach to realize self-calibration of distributed simulations utilizing live data streams was developed for microscopic simulations of vehicular traffic. This approach was evaluated utilizing data collected on sections of roads in the Atlanta metropolitan area.

• Data analytics for real-time prediction of vehicle trajectories. Predictive simulations for the vehicle tracking problem require the prediction of likely future routes of the target vehicle. A data analysis approach and data structures were developed to predict likely future routes of the target vehicle utilizing historical information as well as recent location information of the vehicle in question.

• Algorithms for efficient execution of replicated transportation simulations. The proposed DDDAS approach utilizes multiple replicated simulations to assess likely future locations of the vehicle being tracked. New algorithms were developed to realize efficient replicated simulations of vehicle traffic that exploit the fact that the different runs will have much in common. Techniques to avoid repeating common computations and to eliminate computations that do not impact results needed for decision making were developed. These algorithms were evaluated for typical vehicle simulation scenarios.

• Analysis of approaches to data distribution. Approaches to efficiently distribute data from sensors to the distributed simulation were evaluated. Two approaches to data distribution based on APIs defined in the High Level Architecture standard for distributed simulation were assessed. These analyses considered the energy required to efficiently distribute sensor data for DDDAS usage scenarios.

• Energy analysis of synchronization in distributed simulation. Preliminary work was conducted to experimentally evaluate the energy consumed by synchronization algorithms for distributed simulation and to identify behaviors that significantly impact energy use. To our knowledge, these studies represent the first research to evaluate this issue, potentially opening new avenues of inquiry by the distributed simulation community. These initial studies focused on two conservative synchronization algorithms widely used in the distributed simulation field.

• Parallel algorithms using non-negative matrix factorization for vehicle detection. Exploitation of nonnegative matrix factorization algorithms for detecting vehicles from video traces was examined. An approach to achieve efficient parallel execution of the algorithms was developed and evaluated.

Distribution Statement

This is block 12 on the SF298 form.

Distribution A - Approved for Public Release

Explanation for Distribution Statement

If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.

SF298 Form

Please attach your SF298 form. A blank SF298 can be found here. Please do not password protect or secure the PDF

The maximum file size for an SF298 is 50MB.

SF298 Fujimoto.pdf

Upload the Report Document. File must be a PDF. Please do not password protect or secure the PDF . The maximum file size for the Report Document is 50MB.

Final_Report-v8.pdf

Upload a Report Document, if any. The maximum file size for the Report Document is 50MB.

Archival Publications (published) during reporting period:

1. R. M. Fujimoto, “Parallel and Distributed Simulation,” Winter Simulation Conference, December 2015.

2. A. Tolk, C. D. Combs, R. M. Fujimoto, C. M. Macal, B. L. Nelson, P. Zimmerman, “Do We Need a National Research Agenda for Modeling and Simulation?” Winter Simulation Conference, December 2015.

3. R. M. Fujimoto and A. Biswas, “An Empirical Study of Energy Consumption in Distributed Simulations,” IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, October 2015, best paper award.

4. P. Pecher, M. Hunter, R. M. Fujimoto, “Efficient Execution of Replicated Transportation Simulations with Uncertain Vehicle Trajectories,” 2015 International Conference on Computational Science, Dynamic Data Driven Application Systems and Large-Scale-Big-Data and Large-Scale-Big-Computing Workshop, June 2015.

5. R. M. Fujimoto and A. Biswas, “On Energy Consumption in Distributed Simulations,” Principles of Advanced Discrete Simulation, June 2015.

6. T. A. Wall, M. O. Rodgers, R. M. Fujimoto, M. Hunter, “A Federated Simulation Method for Multi-Modal Transportation Systems: Combining a Discrete Event-Based Logistics Simulator and a Discrete Time Step-Based Traffic Microsimulator,” Transactions of the Society for Modeling and Simulation Intl., Vol. 91, No. 2, pp. 148-163, February 2015.

7. P. Pecher, M. Hunter, R. M. Fujimoto, “Past and Future Trees: Structures for Predicting Vehicle Trajectories in Real-Time,” Winter Simulation Conference, December 2014.

8. L. Yilmaz, S. Taylor, R. M. Fujimoto, F. Darema, “Panel: The Future of Research in Modeling & Simulation,” Winter Simulation Conference, Dec. 2014.

9. R. M. Fujimoto, A. Guin, M. Hunter, H. Park, R. Kannan, G. Kanitkar, M. Milholen, S. Neal, P. Pecher, “A Dynamic Data Driven Application System for Vehicle Tracking,” 2014 International Conference on Computational Science, Dynamic Data Driven Application Systems Workshop, June 2014.

10. S. Neal, G. Kanitkar, R. M. Fujimoto, “Power Consumption of Data Distribution Management for On-Line Simulations,” Principles of Advanced Discrete Simulation, May 2014.

11. W. Suh, M. Hunter, R. M. Fujimoto, “Ad Hoc Distributed Simulation for Transportation System Monitoring and Near-Term Prediction,” Simulation Modeling Practice and Theory, Vol. 41, pp. 1-14, February 2014.

12. Y.-L. Huang, R. M. Fujimoto, C. Alexopoulos, M. Hunter, “On the Transient Response of Open Queueing Networks Using Ad Hoc Distributed Simulations,” Winter Simulation Conference, December 2013.

Changes in research objectives (if any):

N/A

Change in AFOSR Program Manager, if any:

N/A

Extensions granted or milestones slipped, if any:

No-cost extension granted to August 31, 2015

AFOSR LRIR Number

LRIR Title

Reporting Period

Laboratory Task Manager

Program Officer

Research Objectives

Technical Summary

Funding Summary by Cost Category (by FY, $K)

	Starting FY	FY+1	FY+2
Salary			
Equipment/Facilities			
Supplies			
Total			

Report Document

Report Document - Text Analysis

Report Document - Text Analysis

Appendix Documents

2. Thank You

E-mail user

Dec 01, 2015 15:41:15 Success: Email Sent to: fujimoto@cc.gatech.edu