Partitioning algorithms for agent-based crowd simulation

Wang, Yongwei

2011

Wang, Y. (2011). Partitioning algorithms for agent-based crowd simulation. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/46252>

<https://doi.org/10.32657/10356/46252>

NANYANG TECHNOLOGICAL UNIVERSITY

Partitioning Algorithms for Agent-based Crowd Simulation

A Thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of Master of Engineering

By

Wang Yongwei

Supervisor: Cai Wentong

School of Computer Engineering

2011

Abstract

Simulating crowds is a challenging but important problem. There are various methodologies in the literature ranging from macroscopic numerical flow simulations to detailed, microscopic agent simulations. One key issue for all crowd simulations is scalability. Some methods address this issue through abstraction, describing global properties of homogeneous crowds. However, ideally a modeler should be able to simulate large heterogeneous crowds at fine levels of detail. We are attempting to achieve scalability through the application of distributed simulation techniques to agent-based crowd simulation. Generally, for an agent-based crowd simulation that aims to simulate a large number of agents under different scenarios, a proper partitioning algorithm is an essential factor to the performance of the system. However, selecting/developing an appropriate partitioning algorithm for a specific agent-based crowd simulation still remains an issue. One way is to implement distributed agent-based crowd simulations using multiple partitioning algorithms and to compare their performance. This requires multiple distributed simulation experiments, one for each partitioning algorithm. Considering the resources and the effort needed to perform multiple distributed simulation experiments, it is therefore preferred to predict the performance of partitioning algorithms instead. The research reported in this thesis focuses on investigating an effective and efficient approach to evaluate proposed partitioning algorithms for agent-based crowd simulations. A generic framework and an evaluation process are proposed to address the issue of selecting an appropriate partitioning algorithm by simulating the effect of a partitioning algorithm

so as to predict its performance for the agent-based simulations.

Our study also focuses on investigating and developing efficient and effective partitioning algorithms for crowd simulation. Generally, agents form into groups or follow flow patterns in crowd simulation. Exploiting this with an appropriate partitioning algorithm should enable efficient distribution of agents in the crowd simulation. Clustering is a technique used to group similar objects or values together. It can be used to group agents with similar attributes together and eventually to capture the patterns of the crowd. In our study we have investigated methods of partitioning agents onto machines using clustering algorithms. We proposed two cluster-based partitioning algorithms by applying the K-means and grid-based clustering algorithms. The proposed framework and evaluation process are used in the evaluation of the cluster-based partitioning algorithms.

Acknowledgments

Firstly, I would like to render my heartfelt appreciation to my mentor, Prof Cai Wentong from the Nanyang Technological University whose invaluable advice, stimulating suggestions, encouragement, and guidance consistently helped the author throughout his research and report writing.

The author would like to extend his gratitude to Asst Prof Low Yoke Hean Malcolm, Asst Prof Zhou Suiping for their consistent help and advice on my research. Furthermore, thanks also go to the COSMOS project members: Anson Liang Yuanxi, Kabilen Kadirvelen Sornum, Luo Linbo, Michael Harold Lees, and Xiong Muzhou.

Especially, I would like to give my special thanks to my wife Xu Zhuoying whose understanding and support helped the author complete the research as well as the thesis.

Author's Publications

- Y. Wang, W. Cai, M. Y. H. Low, S. Zhou, F. Tian, L. Luo, D. W. S. Ong, and B. D. Hamilton. 2009. A Framework of Evaluating Partitioning Mechanisms for Agent-based Simulation Systems. the 40th IEEE Annual Simulation Symposium (ANSS '09), SpringSim 2009.
- Y. Wang, M. Lees, W. Cai, S. Zhou and M. Y. H. Low. Cluster Based Partitioning for Agent-Based Crowd Simulations. Winter Simulation Conference 2009

Contents

Abstract	i
Acknowledgments	iii
Author's Publications	iv
List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Background	1
1.2 Motivations and Objectives	3
1.2.1 Evaluating Partitioning algorithms for Agent-based Crowd Simu- lation	3
1.2.2 Cluster-based Partitioning algorithm for Agent-based Crowd Sim- ulation	4
1.3 Outline	7
2 Literature Review	9
2.1 Performance Modeling and Prediction for Parallel and Distributed Com- puting Systems	9
2.1.1 Performance Modeling and Prediction for Distributed and Parallel Computing Systems	10
2.1.2 Performance Prediction for Parallel and Distributed Simulation .	14

2.2	Partitioning Approaches for Agent-based Simulations	16
2.3	Partitioning Algorithms for DVE and Agent-based Simulation	20
2.3.1	Static Partitioning	20
2.3.2	Dynamic Partitioning	21
2.4	Partitioning Algorithms for Crowd Simulation	24
2.5	Summary	26
3	A Framework of Evaluating Partitioning Algorithm for Agent-based Simulation System	28
3.1	Introduction	28
3.2	Framework Architecture	29
3.3	Performance Evaluation Process	32
3.4	Validation of the Framework and Performance Evaluation Process	37
3.5	Case Study	41
3.5.1	Partitioning algorithms	41
3.5.2	Benchmark Values	42
3.5.3	Scenario	42
3.5.4	Performance Analysis	43
3.6	Summary	48
4	Cluster Based Partitioning for Agent-based Crowd Simulations	49
4.1	Introduction	49
4.2	Cluster-based partitioning	49
4.2.1	K-means Partitioning	51
4.2.2	Grid-based Partitioning	52
4.3	Experiments	62
4.3.1	Experiments Setup	63

4.3.2	Case Study 1 - Effect of Grouping Degree	64
4.3.3	Case Study 2 - Effect of Moving Pattern	68
4.4	Summary	73
5	Conclusions	75
5.1	Conclusions	75
5.2	Future Work	77

List of Figures

2.1	Performance Prophet Architecture [39]	12
2.2	A Simple Agent-Based System	18
3.1	Architecture of Framework for Evaluating Partitioning Algorithms	30
3.2	Pseudo-code of the Performance Evaluation Process	33
3.3	Analyzer Prediction and Real System Output for Static Algorithm (2 Nodes)	39
3.4	Analyzer Prediction and Real System Output for Static Algorithm (4 Nodes)	40
3.5	Analyzer Prediction and Real System Output for FALB Algorithm (2 Nodes)	40
3.6	Analyzer Prediction and Real System Output for FALB Algorithm (4 Nodes)	41
3.7	Performance under Normal Situation	43
3.8	Performance under Emergency Situation	43
3.9	Load Imbalance Factors among Partitions under Emergency Situation	44
3.10	Step Cost under Emergency Situation	45
3.11	Partitions under Clustered Situation	45
3.12	Communication Cost under Emergency Situation	46
3.13	Percentage of Communication Cost under Emergency Situation	47
4.1	Group Crossing Pattern	50
4.2	Simple Grid-based Partitioning Demonstration	52
4.3	Steps of Grid-based Partitioning Algorithm	56
4.4	Pseudo-code of Grid-based Clustering Algorithm	57

4.5	Interest Management between Individual and Clusters	60
4.6	Interest Management between Clusters	60
4.7	Partitioning Performance for Different Number of Agents	65
4.8	Partitioning Performance for Different Group Percentage	67
4.9	Partitioning Performance for Different Number of Groups	67
4.10	Circular Movement Scenario	69
4.11	Partitioning Performance for Circular Movement	69
4.12	Virtual Environment with Corridor	70
4.13	Partitioning Performance for Different Corridor Widths	71
4.14	Partitioning Performance for Different Corridor Lengths	73

List of Tables

2.1	A Comparison of the Properties of Approaches for the Performance Modeling and Prediction of Parallel and Distributed Computing Systems [42]	14
3.1	Analysis Metrics	37
3.2	Calculated Costs	38
4.1	Parameter space	64
4.2	The Execution Time for Different Partitioning Algorithms (Number of Agents $n = 1000$ & Group % $g = 1$ & Group Num $m = 8$)	66
4.3	Corridor Parameters	71

Chapter 1

Introduction

Crowd simulation has drawn significant research efforts from a variety of fields. The agent-based approach is followed by many research groups for high fidelity crowd simulation. The scalability of the agent-based system is a main issue raised. In order to solve the issue, the parallel/distributed simulation is an obvious solution. So far, several partitioning algorithms are proposed in the area of Distributed Virtual Environment (DVE) ¹ as well as agent-based simulation. It is critical to the performance of the crowd simulation to select/develop a proper partitioning algorithm.

1.1 Background

With the advancement in the high performance parallel/distributed computer systems, the high speed computer graphics, and the high speed networking technologies, crowd simulation that aims to simulate a huge number of individuals in a virtual environment has gathered tremendous momentum in the last few years. Crowd simulations are mainly used in three areas: military training, digital entertainment, and operational study. For the CrOwd Simulation for Military OperationS (COSMOS) project [2], it aims to develop a simulation system for military training as well as the operational study of the normal and emergency situation of public transportation systems, and public gatherings.

¹In terms of partitioning, DVE and agent-based simulation share many similarities (Refer to Section 2.3 for more discussion).

Two modeling approaches for crowd simulation are normally followed by researchers: macroscopic and microscopic approaches. In the macroscopic approach, the model usually treats the behavior of the crowd as a large-scale interactive system. The interaction of every single entity is not handled individually. Statistical and probabilistic assumptions are usually needed in this approach. On the contrary, the microscopic approach controls the crowd at an individual level. It provides a higher degree of variety in terms of the behavior modeling.

Agent-based simulation uses a microscopic approach and has been followed by many research groups [1, 16, 38, 45, 46, 47, 49]. For agent-based simulation, each individual is modeled as an intelligent agent, which may have the capabilities to behave in the simulated world ranging from reacting to triggered events to adapting to a complex dynamic environment. Autonomous individuals, or more precisely agents, are regulated by sets of decision rules and each agent can make independent decisions. The rules are usually based on the local information in the area of interest (AOI) of a single agent. Heterogeneous behaviors of individuals may be achieved by tuning the attribute and behavioral settings of agents and each agent is given the ability to sense, reason, and act according to different situations. An agent-based model is the most natural way to model behaviors with strong individual differentiations and is the most active modeling mechanism used for crowd simulation in the community currently. However, for the application that simulates a large-scale crowd, the computational resources required increase considerably when an agent-based model is used.

In order to make the agent-based system scalable for simulating a large number of individuals, an effective and efficient parallel/distributed approach should be adopted. The distributed approach is considered due to the hardware and software requirements of our ongoing project. The main challenge of this approach is to select/develop a proper partitioning algorithm for a specific scenario.

1.2 Motivations and Objectives

Our research is conducted on the COSMOS and its agent mode is adopted. We assume that the virtual environment is static during simulation in order to focus on partitioning agents in agent-based crowd simulation. If there are dynamic moving objects in the virtual environment, we can treat them as agents which have different working memory, states and intelligence. HLA/RTI is used as underlying infrastructure to build our distributed simulation systems. But our research aims to design a general framework that is independent of the hardware and underlying infrastructure.

1.2.1 Evaluating Partitioning algorithms for Agent-based Crowd Simulation

The partitioning algorithm can seriously affect the overall performance of a distributed agent-based simulation system, since it determines not only the computation load on each server, but also the inter-server communication cost. Several algorithms [27, 48] were proposed, which aim to provide globally optimized partitioning solution for an agent-based simulation by considering both computation load and communication cost. Several heuristic algorithms, such as Genetic Algorithm [18], Ant Colony Systems [12], and Greedy Randomized Adaptive Search [40], were also proposed. Rather than providing a globally optimized solution, these algorithms tend to find a locally optimized partitioning solution with lower computational cost that incurs in executing the partitioning algorithms. However, it is impossible to find a partitioning algorithm that is suitable for all simulation scenarios. To determine a proper partitioning algorithm, one should evaluate the performance of each partitioning algorithm for an agent-based simulation under specific scenarios. Ideally, we can implement a partitioning algorithm and integrate it with an agent-based simulation system to analyze its performance. But it

is not feasible and impractical to implement multiple partitioning algorithms just for evaluating their performance.

Regarding the evaluation of partitioning algorithms, our research focuses on the evaluation of partitioning algorithms virtually by simulating their effects, using an unpartitioned, sequential version of the agent-based simulation system, before the selected partitioning algorithm is eventually implemented in the distributed agent-based simulation system. We also aim to design a standard framework which is independent of the software and hardware configurations of the real distributed agent-based system. It is common that a distributed system will be run on computers with different hardware and/or interconnection software packages. The framework should be generic enough to adapt to this kind of variations.

The proposed evaluation framework should predict the performance of applying a specific partitioning algorithm to a distributed agent-based system by considering the underlying hardware and software configurations. It is necessary to validate the predication against the real distributed system to prove the correctness of the proposed framework. In order to validate the prediction, a real distributed system applying the corresponding partitioning algorithm should be developed and the predicted performance with that of the real distributed system should be compared.

1.2.2 Cluster-based Partitioning algorithm for Agent-based Crowd Simulation

Parallel/distributed simulation is a natural approach to make an agent-based system scalable for simulating a large number of individuals. Parallelism and distribution have been used for many years in simulation as a means of speeding up the execution of programs. The idea is to evenly distribute the program or system among a series of processors such that each processor can execute code in parallel. Different programs

and systems have different degrees of parallelism and the amount of code which can be executed in parallel determines the extent to which it can benefit from parallel execution.

Parallel or distributed execution typically involves periods of parallel execution interleaved with synchronization points. In between the synchronization periods is where the parallel execution occurs and where speed up is gained. The amount of speed up achieved during each of these periods is always determined by the processor which takes longest to execute its share of the load. Therefore, an effective load balancing, which distribute computation amongst processors, is essential to the benefit of the parallel execution.

On the other hand, the complexity and the degree of interaction between the sub-components of the system affects the amount of overhead associated with communication. To reduce the overhead, a proper interest management algorithm should be deployed. Interest management algorithms [8, 30, 31, 34, 35] are concerned with limiting network communication by determining who is interested in which message. The notion of *interested* here is often application dependent but in agent-based simulation and DVE's this is typically associated with the entities' locality. Those entities nearby to each other are considered to be interested in one another. By partitioning entities which interact frequently onto the same computing node, communication overhead will be reduced.

The load balancing and interest management are the two main factors that determine the effectiveness of a parallel and distributed simulation. Most of the time, these two factors are inter-related. The partitioning problem has been studied in the Distributed Virtual Environment (DVE) and Massively Multi-player Online Role-playing Game (MMORPG) communities. The problem begins with a distributed collection of processors which when put together form a single virtual environment (or computational space). However, the challenge lies in partitioning the environment amongst the processors, while still maintaining awareness and consistency. When any system is decomposed and split amongst a set of processors, in order to achieve the same results it is necessary

to ensure the entire system remains consistent. However, it is not a requirement that each processor has a globally consistent view of the entire environment. It is this fact which can be exploited when reducing the communication between processors, the challenge is then how to ensure only the minimal set of necessary information is communicated between processors. The problem we investigate is how to evenly *partition* the agents onto processors to minimize inter-processor communication in agent-based simulation.

Migration is another main factor that determines the effectiveness of a parallel and distributed simulation. In our study, we differentiate between communication and migration. Typically, migrating an agent between two processors involves deleting its state in the current processor, transferring its state (e.g., memory, and experience), and initializing a new agent in the other processor. As a result, it requires more computational power and data transfer for migrating an agent's state than that of retrieving an agent's state to maintain consistence, especially when a complex agent model with large working memory is adopted.

In crowd simulations, crowds have been shown to naturally form groups and flow patterns [6]. Several individuals may share the same goals and move together for a period of time during simulation. These individuals can be easily classified into groups and flow patterns. By exploiting this with an appropriate partitioning algorithm, it will enable efficient distribution of agent in the crowd simulation. Regarding the partitioning algorithm itself, our research focuses on investigating the application of clustering techniques to partition crowd simulations.

Clustering is a technique used to group similar objects or values together. It has drawn significant research efforts in machine learning, data mining, pattern recognition, image processing and bioinformatics. In general, the existing clustering algorithms can be classified into four categories: partitioning based, hierarchical-based, density-based, and grid-based methods [17]. K-means is a partitioning based clustering algorithm which

groups objects or values into k clusters based on some attributes which are assumed to form a vector space. K-means has been applied in a wide variety of disciplines, including Sociology, Geography and Computer Science. In our study, we consider the agents positions as attributes which span a 2D Euclidean vector space, that is, the environment. K-means clusters agents near each other into one cluster. For distributed crowd simulation, agents interact with nearby agents frequently. By applying k-means clustering algorithm to partition the crowd simulation, we may be able to maintain locality and reduce communications. Compared with other clustering methods, the grid-based clustering is known for its fast processing time [24]. The algorithm also offers a great deal of control as the execution time is proportional to the defined grid size rather than the data objects. This algorithm uses a single uniform grid mesh to partition the entire problem domain into cells. Statistics are collected regarding the data objects (in our case agents) located within a cell to form a summary of the cell's content. Clustering is then performed on the grid cells based on this statistical summary of each cell. In our study, we seek to utilize the grid-based clustering algorithm to cluster agents based on their location (cells) and their long term goals (collected statistics).

We will evaluate and analyze the performance of the k-means and grid-based clustering algorithms for crowd simulation with the proposed evaluation framework.

1.3 Outline

The report is organized as follows. Chapter one introduces the background, the motivations and the objectives of the research. Chapter two focuses on the literature review of performance prediction for parallel/distributed systems, common partitioning approaches for agent-based simulation, general partitioning algorithms for DVEs as well as agent-based simulation, and partitioning algorithm for agent-based crowd simulation. Chapter

three presents the proposed framework of evaluating partitioning algorithms for agent-based simulation systems. Chapter 4 presents our proposed cluster-based partitioning method for crowd simulation. Finally, chapter five concludes our research and summarize our contributions, and states the possible extension of our research.

Chapter 2

Literature Review

The objectives of the author’s research focus on evaluating partitioning algorithms for agent-based simulation, validating the evaluation process and framework, and applying clustering techniques to partitioning agents in agent-based crowd simulation. This chapter reviews the necessary underlying techniques to achieve these objectives. Firstly, we review the approaches to performance modeling and prediction of parallel and distributed computing systems, and performance prediction for distributed simulation in particular. After that, we present a classification of partitioning approaches for agent-based simulation based on how the environment is maintained and how agents are distributed among processors. We also review the existing partitioning algorithms for distributed virtual environments, agent-based simulation, and particle simulation. Finally, we survey the state-of-the-art of partitioning algorithms for agent-based crowd simulation.

2.1 Performance Modeling and Prediction for Parallel and Distributed Computing Systems

Performance is one of the key factors of parallel and distributed computing systems. A significant effort has been devoted to the development of approaches for performance modeling and prediction of parallel and distributed computing systems in the past 20 years. The commonly used techniques for performance evaluation of computing systems

include: measurement, mathematical modeling, and simulation. Measurement techniques require that the system under study is available for experimentation, and their applicability is limited to existing systems. Mathematical performance models usually lack the system's structural information, and therefore, are not suitable for detailed performance analysis. The detailed performance analysis often involves the modification of structure of the model to reflect system structural changes, and the simulation of what would be the performance of the system under study if its structure is changed. Simulation models demand large computational and storage resources, and their execution may be slow. Hybrid approaches of these three techniques are also possible. In the next subsection, a few approaches are introduced, all of which are hybrids of these three techniques.

2.1.1 Performance Modeling and Prediction for Distributed and Parallel Computing Systems

Several approaches have been proposed and studied by researchers for performance modeling and prediction of parallel and distributed computing systems. The most related approaches are introduced in the following paragraphs.

2.1.1.1 Performance and Architecture Laboratory's (PAL) Approach [20]:

In [20], Kerbyson, Alme et al. presented a predictive analytical model that encompasses the performance and scaling characteristics of an important ASCI application. The research was conducted on SAGE (SAIC's Adaptive Grid Eulerian hydro-code), which is a multidimensional (1D, 2D, and 3D), multi-material, Eulerian hydrodynamics code with adaptive mesh refinement. The performance prediction model is formulated by analysis of the code, inspection of key data structures, and analysis of traces gathered at run-time. They divided an iteration cycle into three non-overlapping operations:

- one or more gather operations to obtain a copy of the local and remote neighbor data

- computation in each of the local cells
- one or more scatter operations to update data on remote processors.

They estimated the performance by means of the equation below:

$$T_{cycle}(P, E) = T_{comp}(E) + T_{memcon}(P, E) + T_{GScomm}(P, E) + T_{allreduce}(P) \quad (\text{Eq. 2.1})$$

where

P is the number of processors

E is the number of cells per processors

$T_{comp}(E)$ is the computation time

$T_{GScomm}(P, E)$ is the gather and scatter communication time

$T_{allreduce}(P)$ is the allreduce communication time

$T_{memcon}(P, E)$ is memory contention that may occur between processors

The main factors are $T_{comp}(E)$ and $T_{GScomm}(P, E)$ and they assume the operations are non-overlapping. They also developed mathematical models to estimate the costs based on some basic performance measurements of SAGE.

They made comparison of predictions and measurements on four systems. The predictions from the performance model show high accuracy, mostly within 10% of the actual measurements.

This approach is a hybrid of mathematic modeling and measurement. The $T_{comp}(E)$ is measured from an execution of SAGE on a single PE for a given number of cells E. Other operations costs are also estimated as functions of measured metrics of SAGE.

2.1.1.2 Performance Prophet [39]:

Figure 2.1 depicts the architecture of Performance Prophet. The main components of Performance Prophet are Teuta and Performance Estimator. Teuta is a platform independent tool for UML-based modeling of parallel and distributed programs. The model

Figure 2.1: Performance Prophet Architecture [39]

of the distributed and parallel program (i.e., workload model) is composed from the predefined building blocks (i.e., workload elements). The implementation of Performance Estimator is based on CSIM [44]. CSIM is a general purpose simulation engine, which offers a set of generic classes that can be used to compose discrete event simulation models. The communication between Teuta and the Performance Estimator is done via Data Repository.

The design of Performance Prophet enables the model based performance analysis of parallel and distributed programs. The user develops graphically the performance model of a program based on the UML by using Teuta. The program model is composed of existing building blocks; therefore, it can be used to automatically transform from UML into a C++ representation. The Performance Estimator evaluates the performance of the program on the computer architecture selected by the user. In this way, it is possible to experiment with the model rather than with the real program.

They validated the simulation model by comparing the simulation results with the measurement results, and found that their simulation model can provide performance prediction results with sufficient accuracy.

The performance prophet is also a hybrid approach. For instance, they split the behavior of computing into action states and waiting states. A mathematical model is used to model the performance behavior of action states; whereas the performance behavior of waiting states is simulated.

2.1.1.3 POEMS [5] [4]:

Performance Oriented End-to-end Modeling System (POEMS) aims to create and experimentally evaluate a problem solving environment for end-to-end performance modeling of complex parallel/distributed systems, application software, runtime and operating system software, and hardware architecture. It proposes a methodology for the evaluation of system model using multiple evaluation tools. The model of system was composed of component models. They defined three types of component models, namely, multi-scale models, multi-domain models, and multi-paradigm models. Multi-scale models allow different components of a system to be modeled at varying levels of detail via the use of associative-object-based module interfaces. Multi-domain models integrate components from three different semantic domains, namely application software, runtime/OS software, and hardware. Multi-paradigm models allow components to be modeled using the model evaluation paradigm (analytical, simulation, or the software or hardware system itself) that is most appropriate for the goals of the performance study.

Since this approach requires different levels and different kinds of component models, the inter-operability between those component models maybe a problem if the output of the evaluation tool of a component model is not interpretable with that of the subsequent component models.

2.1.1.4 Comparison:

In [42], Sabri, Ivona et al. did a survey on the performance modeling and prediction of distributed and parallel computing systems. They evaluated the approaches using four

metrics:

- *Model development effort* indicates the endeavor of the user of a specific approach for building the model.
- *Model evaluation effort* indicates the time and the computing resources that are needed to evaluate the model.
- *Efficiency of the approach* considers the effort needed to develop and evaluate the model. For instance, the efficiency of an approach is high if the model development and evaluation effort is low.
- *Accuracy* indicates the exactness of an approach. The estimation of the accuracy is commonly performed by a comparison of prediction results with measurement results.

Table 2.1: A Comparison of the Properties of Approaches for the Performance Modeling and Prediction of Parallel and Distributed Computing Systems [42]

Approach	Mod.Dev.	Mod.Eval.	Efficiency	Accuracy
PAL	High	Low	Medium	Medium
Prophet	Medium	Low	High	Medium
POEMS	Medium	Medium	Medium	Medium

In Table 2.1, they made a comparison of several approaches to performance modeling and prediction of distributed and parallel computing systems. All of three approaches have a medium accuracy, while Prophet has the highest efficiency.

2.1.2 Performance Prediction for Parallel and Distributed Simulation

In [13], a performance estimation tool N-MAP was developed, which estimates the performance of a parallel program from a parallel code skeleton written by the user. The

specifications are used as means for roughly expressing the skeletal structure of a parallel application and its preliminary performance prediction. N-MAP then parses and distributes the specifications to parallel virtual processors. The virtual processors execute the specifications and log the execution trace. The trace will be used for view and analysis. As an example of how the tool can be used to estimate the parallel performance of a simplified simulation model, the authors wrote a petri net simulation code which implements variations of both conservative and optimistic protocols. The limitation of their approach is that they rely on code skeleton to analyze and predict the performance. Given a segment of code, different input parameters may result in different execution time of the segment. This behavior is difficult to be characterized using code skeleton. In addition, it is also tedious to generate a code skeleton for a large and complex program.

In [25, 26], a study was conducted on the prediction of the performance for parallel discrete event simulation. It focuses on predicting the performance of parallel simulation using sequential simulation executions. In their work, they developed different performance analyzer tools for both synchronous and asynchronous conservative protocols. They also implemented the performance analyzer tool as an independent module from the implementation of the sequential simulation. As a result, the tool works as a plug-in tool to a sequential simulation. The performance analyzer tool can help analyze the speedup of parallelization as the sequential simulation runs. We follow the same idea of predicting the parallel and distributed performance. We also follow their idea to develop an independent analyzer module that can adapt to different agent-based simulations. Their study aims to investigate the effectiveness of parallelizing sequential discrete events simulation using different protocols. However, our study focuses on evaluating the performance of partitioning algorithms for agent-based simulation under the super-step conservative protocol.

In [51], Xu and Chung developed a model to predict the performance of synchronous discrete event simulation. Their model considers the two most important factors for the

performance of synchronous simulation: load balancing and communication. The effect of load balancing in a synchronous simulation is computed using probability distribution models. They derived a formula that computes the cost of synchronous simulation considering communication overhead and computation granularity. Even though the formula is simple, they claim that it is effective in capturing the most important factors for the synchronous simulation. The formula helps them to predict the maximum speed up achievable by synchronous simulation. In order to examine the prediction model, they have simulated several large ISCAS logic circuits and a simple PCS network simulation on an SGI Origin 2000 and Terascale Computing System (TCS) at the Pittsburgh Super-computing Center. The results of the experiment show that their performance model accurately predicts the performance of synchronous simulation. The performance model developed is used to analyze the effect of several factors that may improve the performance of synchronous simulation. The factors include problem size, load balancing, computation granularity, communication overhead, and partitioning. However, their approach cannot be applied directly to predict the performance for agent-based simulation. Logic circuits and networks are static and do not change duration simulation. The performance prediction for agent-based simulation is far more complex, considering its dynamic nature. But, the two factors, load balancing and communication, identified by them can be used in evaluating agent-based simulation because these two factors are essential to the performance of distributed agent-based simulation as well.

2.2 Partitioning Approaches for Agent-based Simulations

An agent-based simulation normally consists of a number of agents and a virtual environment with which agents can sense, perceive and act. Furthermore, agents generally

interact with other agents in the virtual environment. Partitioning an agent-based simulation requires not only partitioning the virtual environment but also assigning agents to servers. These two tasks can be treated together or independently. Based on how these two tasks are handled, we defined four general partitioning approaches for the agent-based simulation: *centralized virtual environment*, *replicated virtual environment*, *partition virtual environment with agents*, and *partition virtual environment independently*. We will evaluate the advantages and disadvantages of these four approaches in three dimensions, namely *consistency*, *load balance* and *communication cost*. Consistency refers to the fact that an object or an agent in the environment is seen by the surrounded agents or cameras with the same state. This is important for simulating a large crowd in a realistic manner, especially when interactions between agents and environment are frequent. Load balance is a performance measure that indicates whether the distributed servers are efficiently utilized. Overloaded servers will lower the performance of the system, while lightly loaded servers indicate starvation. Communication cost refers to the time spent on transferring information from one server to other servers. Since the servers are linked together by LAN or Internet, it takes time to transfer data and process data. Especially when a large crowd is considered, the data transferred can be large, therefore the transferring and processing time will greatly influence the performance of the system. Figure 2.2 shows a simple agent-based system with eight agents in an open space. We will use it to demonstrate the different approaches.

Centralized Virtual Environment: In [43], it uses centralized server architecture for tracking and tracing of simulated agents using a centralized environment model. In this approach, as shown in Figure 2.2, one central server is allocated to maintain virtual environment while the agents are distributed to four client servers based on their positions. The interactions between environment and agents are handled by the central server, so that the consistency can be easily maintained. But the bottleneck lies on the

Figure 2.2: A Simple Agent-Based System

central server. When a large crowd is simulated, the communications between the central server and the distributed clients, and the computation load of the central server grow dramatically. This approach is not scalable for simulating a large number of entities.

Replicated Virtual Environment: For supporting real-time interaction in distributed virtual environments, it is common to replicate virtual world data at clients from the server [37]. It is also applicable for agent-based simulation especially when the virtual environments are static. In this approach, the eight agents can be distributed into four servers based on their positions; while virtual environment is replicated among the four servers so that each server has a full view of the virtual environment. The communication cost incurred when an agent queries the virtual environment is eliminated because the whole virtual environment is replicated in the server that the agent resides. Dynamic

change in the virtual environment is broadcast among servers which will incur communication cost as well as computation cost. So this approach is only suitable for static virtual environment or the virtual environment with infrequent change.

Partition Virtual Environment with Agents: In this approach, the virtual environment is divided into four parts and the eight agents are divided accordingly as shown in Figure 2.2. This approach is adopted by most of the current distributed simulation systems (for example [7]). The communication cost is reduced since the local states are maintained in local servers compared to the first approach. Each distributed server handles only its local agents such that there is no computation bottleneck as long as the partition algorithm is effective and efficient. Currently, there are several existing partition algorithms (e.g., GRASP, genetic, ant colony system) that can be used directly. However, execution of these algorithm may take considerable time, especially when the environment is crowded. Consistency is also a drawback of this approach. Comparing with the centralized virtual environment approach, additional communication cost, which refers to the cost of broadcasting dynamic changes of the virtual environment in one server to other servers, will be incurred to maintain the consistency.

Partition Virtual Environment Independently: Agents can be partitioned by physical position, group or other attributes. The virtual environment can be partitioned independently such that there are two groups of servers. One group of servers handles virtual environment; while the other group of servers handles agents. For example, in Figure 2.2, we can partition the eight agents into four groups by their group attributes while virtual environment is partitioned into four regions. In this approach, the virtual environment consistency is maintained among the environment servers without considering the agents. It reduces the overhead time incurred in the last approach to maintain the virtual environment consistency. Since agents and virtual environment are independent, different partitioning algorithms can be used for each to achieve better performance. The

drawback of this approach lies on the computation and communication overhead. When an agent queries the virtual environment, the virtual environment servers will spend overhead time to allocate the query to a proper server.

After a certain partitioning approach is selected for an agent-based simulation, the next step is to select a suitable partitioning algorithm to formulate and to maintain the partitions. In certain partitioning approaches, it is possible to apply different partitioning algorithms to distribute the agents and partition the virtual environment for an agent-based simulation.

2.3 Partitioning Algorithms for DVE and Agent-based Simulation

Partitioning algorithms have been drawn significant research efforts in many research fields, especially in the area of DVEs and distributed agent-based systems (multi-agent systems). The problem of partitioning DVEs is similar to partitioning agent-based simulations. Generally there are characters moving and interacting within an environment and the problem involves assigning avatars or agents to servers. Partitioning algorithms can be roughly categorized into two classes: static partitioning and dynamic partitioning.

2.3.1 Static Partitioning

Many distributed agent-based/virtual environment systems adopt the static environment decomposition (for example [7, 22, 23]). During initialization, each server is assigned a part of the virtual environment under control. We can divide the virtual environment evenly or based on the prior knowledge of the scenarios among partitions. The divisions of the virtual environment are fixed duration the simulation. When agents/objects move around in the virtual environment, migration between partitions will incur. Communication can incur when it is necessary to retrieve data from other partition(s). In some

agent systems this may be sensible as the movement of the agents forms no particular pattern or is very restricted. In many cases, however, this approach is mainly adopted for simplicity. The suitability of a given partitioning algorithm depends on many characteristics of an agent-based simulation (e.g., the density of agents, movement of the agents, and distribution of agents). A dynamic approach, which can minimize the load imbalance, communications and migrations among different partitions during simulation, should provide a good solution in those cases where agents move dynamically in the virtual environment.

2.3.2 Dynamic Partitioning

In some agent-based simulations, for example the evacuation simulation, individuals may move and gather in specific areas (e.g., exit area). This makes the static partitioning algorithm inefficient. Dynamic partitioning tends to adapt the dynamic change in agent-based simulation, DVEs and Massively Multi-player Online Role-playing Game (MMPOGs) by balancing computation loads and communication costs. It can be triggered either at a predefined fixed time interval or by some predefined events, such as load imbalance, communication imbalance, etc. For the first case, the time interval affects the system performance. If the time interval is set too small, unnecessary repartitions will be triggered; while if it is set too big, the system cannot detect the imbalance timely. For the second case, the state, such as load and communications, of each server should be checked frequently. Several research works have been done on the dynamic partitioning in the area of DVEs and they will be briefly described in the next few paragraphs

In [27], Lui and Chan provided a heuristic partitioning algorithm. The problem of partitioning is treated as a problem of solving a weighted graph in their research. The nodes represent the avatars and edges represent communication links between any pair of avatars. The weight of an edge is the size of message to be communicated between

two avatars in byte. The communication cost between two partitions is defined as the summation of weight of the cross-border edges from one partition to the other partition. Their algorithm tries to find a minimum C_p :

$$C_p = W_1 C_p^W + W_2 C_p^L$$

with $W_1 + W_2 = 1$ (Eq. 2.2)

Where C_p^W and C_p^L are the computational workload cost and the communication cost respectively. W_1 and W_2 represent the relative importance of the computational workload cost and the communication cost respectively. They can be tuned based on communication links and computer configurations.

Their algorithm has three key components: recursive bisection partitioning algorithm, layering partitioning algorithm, and communication refinement partitioning algorithm. The main concept of the recursive bisection partitioning algorithm is to divide the avatars in the virtual environment into groups and then, based on the divide-and-conquer technique to find an initial partitioning strategy. The layering partitioning algorithm labels each avatar using a partition number. The label serves as a possibility of mapping the corresponding avatar to the partition with the same number. Linear programming optimization is used to decide whether the labeled avatar should be remapped. It is used to reduce the overall computational workload cost of the system by balancing the load among partitions. The communication refinement partitioning algorithm is employed to reassign some avatars to other partitions so as to reduce the server-to-server communication cost. Their optimization is conducted on graph which is popularly adopted in the study of the distributed virtual environment. In conclusion, their approach considers two important measurements of distributed systems: communication cost and computational workload cost. In our research, we consider the same cost when predicting the performance of a partitioning algorithm.

The idea of dynamically applying a heuristic search method that provides a good assignment of clients to servers as the state of the DVE system changes has been followed by many researchers. In [32] an analysis of various approaches to the problem of partitioning in DVEs are presented, including simulated annealing, random search, ant colony system, genetic algorithms as well as the greedy randomized adaptive search. Among these algorithms, the author claims that the ant colony heuristic is shown to give the best performance and overhead trade-off for large DVEs. The ant colony algorithm is based on the system of pheromone that ant colonies use for locating food. Each ant intermittently places the pheromone along the path which it follows. Subsequent ants will then follow paths with high pheromone concentrations. The algorithm defines border avatars as those avatars which reside on a different server to other avatars within their Area of Influence (AOI). A virtual ant colony is then used to search a set of candidate servers for all the border avatars.

In [36] an approach is presented which considers agents access patterns to the shared state as a means of partitioning agents onto processors. The common state of the environment is stored on processors that are different from where the agents are. To reduce communication cost the state is moved closer to the agents which access the same state. This approach uses the concept of Spheres of Influence (SoI) to partition agents, meaning agents which interact with the same area of the environment are also close in terms of the system infrastructure. A second assumption is that the SoIs of agents do not change drastically over short time periods, but instead change gradually over the course of the simulation. To prevent the SoIs changing dramatically, the environment must be large in comparison to the velocity at which the agents operate, i.e., an agent cannot move from one side of the environment to the other side in a few time steps.

2.4 Partitioning Algorithms for Crowd Simulation

Crowd simulation with a large number of autonomous agents with realistic behavior is still a challenge in computing. Parallel and distributed computing system should be employed to solve this problem. However, it introduces another challenge. How can we reasonably allocate the workload in parallel and distributed systems to achieve better performance? This challenge has draw significant research effort in parallel and distributed computing. Typically, there are two different approaches for partitioning a crowd simulation. One approach is region-based. Virtual environment is divided into regions and all agents in a specific region are assigned to a computation node. The other approach is based on the criterion of workload. Agents are grouped according to some criteria and are assigned to a specific computation node which handle that group.

In [50], Viguera, Lozano and et al. investigated regional-based partitioning methods for crowd simulation and compared their performance. They define a fitness function to evaluate the performance of a partitioning method:

$$H(P) = \omega_1 * \alpha(P) + \omega_2 * \beta(P), \quad \omega_1 + \omega_2 = 1 \quad (\text{Eq. 2.3})$$

$\alpha(P)$ is computed as the sum of all the agents whose AOIs intersect two or more regions of the virtual environment; while $\beta(P)$ is computed as the standard deviation of the average number of agents that each region contains. ω_1 and ω_2 are weighting factors between 0 and 1. The fitness function aims to measure the number of border agents with their Area of Interest (AOI) crossing the region borders and load imbalance among partitions. They claimed that a partitioning method with minimum $H(P)$ will provide the best performance for crowd simulation. They implemented three regional-based partitioning methods by using R-tree, genetic algorithm and convex hull. The R-tree and genetic algorithm adaptively partition the virtual environment into overlapped rectangle regions; while the convex hull adaptively partitions the virtual environment into regions with

irregular shapes. They evaluated the performance of different partitioning methods in terms of the fitness function and execution time and claimed that the convex hull method outperforms both R-tree and genetic algorithm in both terms. However, they defined the fitness function for each static state during crowd simulation and failed to consider that agents may migrate from one partition to another between two static states. For crowd simulation, it is highly likely that agents may cross the region borders no matter what the partitioning method is employed. Furthermore, comparing with retrieving an agent's state to maintain consistency, it requires more communication and computation overhead for migrating an agent from one partition to another especially when a complex agent model with large working memory is adopted. As a result, it is essential that we should reduce the number of agents that are migrated from one partition to another partition. At the same time, we should also reduce the number of border agents and balance the number of agents in each partition.

The approach to group agents according to certain criteria and assign group to a computation node has not been drawn significant research effort in crowd simulation. It fails to consider an essential factor, which is interest management, that may affect the performance of a partitioning method for crowd simulation. Agents in crowd simulation interact with nearby agents and environment continuously. Maintaining locality is crucial to an effective partitioning method for crowd simulation.

In our study, we utilize crowd pattern to partition the crowd simulation by applying clustering techniques. We aim to balance the computational load, minimize the communications and migrations at the same time.

Crowds have been shown to naturally form groups and flow patterns [6]. Exploiting this with an appropriate partitioning algorithm should enable efficient distribution of crowd simulation. In [52], the K-means clustering algorithm was used to group objects into k clusters based on their positions. Traditional K-means give us tightly localized,

convex boundaries which favor reduction of both computational cost and communication cost. But it cannot guarantee a load balanced partitioning. Some researchers introduced another parameter to the traditional K-means algorithm to tune the effect of load-balance. If a partition is overloaded, the parameter tends to contract the partition.

K-means approach has a few drawbacks. Firstly, they need a central server to process the K-means algorithm. Secondly, since the number of servers are predefined, the number of clusters can be identified by K-means is fixed. However, the number of pattern groups can be varied in different scenarios or at different simulation stage of a specific scenario. In this case, the K-means may not be able to identify the clusters correctly. Thirdly, this approach cannot differentiate individuals from pattern groups.

2.5 Summary

Significant research efforts have been devoted to predicate and evaluate parallel and distributed computing systems. There are three common main techniques for performance modeling and prediction: measurement, mathematical modeling and simulation. We reviewed three hybrid approaches on predicating and evaluating parallel and distributed computing system. These three approaches demonstrate that accurate predications of parallel and distributed computing systems are achievable. In our study, we follow a hybrid approach which combines these three techniques. Measurement is used to make our evaluation framework adaptive to different software and hardware configurations. A mathematical model is used to quantify different metrics in order to make comparisons of the performance among different partitioning algorithms. And simulation is used to simulate the partitioning algorithms in a non-distributed system rather than deploying distributed simulation systems for all partitioning algorithms under evaluation.

Studies have been done on evaluating parallel and distributed simulation. N-MAP adopts a code skeleton approach. The limitation of their approach is the difficulty of

developing a code skeleton for a large and complex program. Other research works have been done on evaluating the discrete event simulation. Computation load and communication overload are the two main factors identified for the evaluation of parallel and distributed simulation. In agent based simulation, agent may move from one partition to another partition. The migration cost, therefore, should also be considered. We aim to create an evaluation framework that is dedicated to agent-based simulation, considering the computation load, communication overhead, and migration cost.

Partitioning algorithms for agent-based simulation can be categorized into four approaches: centralized virtual environment, replicated virtual environment, partition virtual environment with agents, and partition virtual environment independently. In our study, the virtual environment is static during simulation. As a result, the replicated virtual environment approach is a natural choice. After deciding partitioning approach, a suitable partitioning algorithm should be selected. Many studies have been done on partitioning algorithms for DVEs and agent-based simulations in the last few years. Most of the studies focus on optimizing a certain cost function. The parameters of the cost function normally include computation load and communication overhead. These partitioning algorithms tend to partition DVEs or agent-based simulation in such a way that the cost function is minimized for each static state. However, agents and objects could move dynamically in the virtual environment in agent-based simulation and DVEs. Migrations of agents and objects should be considered. For agent-based crowd simulation, agents have been shown to naturally form groups and flow patterns. Several agents may share the same goals and move together for a period of time during simulation. The long term interest (i.e., agents' goals) are an important factor for an effective partitioning algorithm. Clustering is a technique used to group similar objects or values together. In our study, we aim to apply the clustering techniques to capture the movement pattern and develop a partitioning algorithm with consideration of computation load, communication overhead, migration cost, and agents' long term interest.

Chapter 3

A Framework of Evaluating Partitioning Algorithm for Agent-based Simulation System

3.1 Introduction

As introduced in Chapter 1, many research works have followed the agent-based approach in crowd simulation. In order to make the agent-based system scalable for simulating a large number of individuals, an effective and efficient parallel/distributed approach should be adopted.

As discussed in chapter 2, in [25, 26], a study was conducted on the prediction of the performance for parallel discrete-event simulation. It focused on analyzing event dependencies and thus predicating the performance of parallel simulation using sequential simulation experiments. We follow the same idea of predicting the parallel and distributed performance. The performance data, such as number of agents, number of messages transferred, and number of migrations, will be logged and processed at each simulation step. At the end of the simulation, our performance analyzer tool will predict the performance for distributed simulation by utilizing the processed data at each simulation time step. The log will be saved to assist us to further analyze the performance of a specific partitioning algorithm. We also follow their idea to develop an independent

analyzer module that can adapt to different agent-based simulations. In [51], a model was developed, which considered the two most important factors for the performance of synchronous simulation, load balancing and communication, in the prediction of the performance. Given the parallel simulation's specifications, the model can predict the performance of synchronous parallel simulation's execution using a sequential simulation system. The authors made a comparison of the predicted performance with the real system performance to prove the correctness of their model. However, their studies focused on the simulation of the computer networks rather than the agent-based simulation systems. Our research follows the idea of the previous studies to predict the performance of a synchronous distributed simulation execution, given an undistributed sequential agent-based simulation system. Considering the unique characteristics of the agent-based simulation system, we propose a generic framework and an evaluation process to evaluate various partitioning algorithms for the system.

3.2 Framework Architecture

The framework aims to evaluate various partitioning algorithms for different simulated scenarios so that an appropriate partitioning algorithm can be selected for each scenario. Generally speaking, we can implement each partitioning algorithm and integrate it into the unpartitioned simulation system to evaluate the performance. But, to implement a distributed simulation, the existing unpartitioned simulation system usually needs to be restructured in order to accommodate the partitioning algorithm. In addition, there are many existing partitioning algorithms to be considered for each scenario, so it is not feasible to implement all of these algorithms with the actual simulation system to evaluate their performance. The proposed framework simulates partition algorithms instead of implementing the real distributed system. The main task is to implement the structures of the partitioning algorithms which maintain the configuration of the partitions. Using

the data collected during the simulation execution, the performance of the partitioning algorithms can then be evaluated.

The framework contains four main components: scenario definition, simulator, partitioning algorithms, and analyzer. The architecture is shown in Figure 3.1 and its main components are described below.

Figure 3.1: Architecture of Framework for Evaluating Partitioning Algorithms

Scenario Definition: The scenario definition is a high level description of the scenario to be simulated by the simulator, such as the number and composition of the agents, the configuration of the virtual environment, and the description of the situation in the virtual environment etc. The information is used by the simulator to initialize the agents and the virtual environment.

Simulator: The simulator is a sequential agent-based simulation system in this framework. It simulates the scenario defined by Scenario Definition. At each simulation

step, the simulation states are logged by the simulator and retrieved by the Analyzer. The simulation states consist of the essential data, such as agents' positions, agents' sensing ranges etc, which are used by partitioning algorithms to partition the agent-based simulation system. The Analyzer maintains the simulation states and shares them among its instances. The instances evaluate the performance of the various partitioning algorithms using the logged simulation states. In our proposed framework, several partition algorithms can be analyzed in one run, because no matter what partitioning algorithm is used, the agents simulated will not be affected by the algorithm.

Partitioning Algorithm: This component implements the structure of each partition algorithm. Instead of partitioning and distributing agents and/or virtual environment to several different servers, it simulates the effect of the real partitioning by maintaining the configuration of each partition under a specific partitioning algorithm. The configuration of each partition is defined by the area and the agents that belong to the partition. An analyzer instance is dedicated to each partitioning algorithm to analyze the performance. It will monitor the configuration of each partition under the specific partitioning algorithm. The framework is designed in such a way that any partitioning algorithm can be injected into it directly. For example, in our case study, in addition to the static partitioning and grid-generation based adaptive partitioning methods, the fast adaptive load balancing method (FALB), which is a modified version of the grid-generation based adaptive load balancing method [11], and static partitioning method are implemented and injected into the framework. To add a new partition algorithm, a new analyzer instance needs to be instantiated. Several other dynamic partitioning algorithms, such as finite element method [21], boundary element method [19] and cluster-based partitioning could also be considered.

Analyzer: It is the main component of the framework. It connects the unpartitioned simulation system, which is the considered Simulator, with the partitioning algorithms

and analyzes the performance with the aid of the benchmark values. Three benchmark values are considered in this framework: communication benchmark value, computation benchmark value and migration benchmark value. They are collected by setting up tests given the hardware and software configurations of the real distributed simulation system to be implemented. The simulator collects the simulation states and the Analyzer shares the simulation states among its instances. The instances combine these simulation states with the benchmark values to generate the cost of the partitioning algorithm at each simulation step. (The details of the evaluation process will be illustrated in the next section.) For each partitioning algorithm under evaluation, an analyzer instance is created to control the process of partitioning. The configuration of a partition under a specific partitioning algorithm is maintained and updated during simulation by the analyzer instance. During the simulation, performance data for each partition, such as the number of agents in the partition, the number of queries for surrounding agents which reside in another partition, and the number of migrations are generated by each analyzer based on the maintained configuration of that partition.

3.3 Performance Evaluation Process

The pseudo-code shown in Figure 3.2 shows the performance evaluation process. There are three main stages in the performance evaluation process, namely initialization, step performance evaluation, and overall performance evaluation. In the initialization stage, all the components, such as the simulator, the analyzer instances, and the partitioning algorithms, are initialized sequentially in a predefined order. After initialization, the main loop of the simulation executes. The simulator advances simulation time per loop to update the simulated agents and/or virtual environment. During a simulation step, the Analyzer will firstly retrieve the simulation states from the simulator. By combining the benchmark values and the simulation states, each analyzer instance evaluates the

```
Run scenario definition to generate the initialization data;
Initialize the simulator;
Initialize the analyzer( $A1, A2, \dots, An$ ) for each partitioning algorithm;
/*Initialize the predefined configuration of each partition*/
Initialize the partition algorithms;
/*The main simulation loop*/
while(timeStamp < EndSimulationTime) do
    simulate();
    for (all analyzers)
        /*step performance*/
        retrievePerformanceData();
        analyzePerformance();
        logEvaluationResult(timeStamp);
        updatePartitionConfiguration();
    endfor
    /*make simulator advance its timeStamp*/
    advanceTime();
endwhile
for (all analyzers)
    /*overall performance*/
    calculate the overall performance;
endfor
```

Figure 3.2: Pseudo-code of the Performance Evaluation Process

step performance under the specified partitioning algorithm in the form of the step cost. The step cost is logged at each simulation step until the simulation ends. In the final stage, the step cost at each simulation step is used to evaluate the overall performance, which is also referred to as the total cost, of the specified partitioning algorithm. Step cost and total cost represent in terms of the time that it takes to complete one simulation step or one simulated scenario respectively.

Three metrics are considered during evaluation, namely communication cost, computation cost and migration cost. Communication cost refers to the time taken when a server retrieves the states of relevant agents from other servers. Computation cost refers to the time taken when an agent performs an update cycle at each simulation step. Migration cost is defined as the time taken when the control of an agent is transferred to another server. The total cost for an agent at a simulation step is the summation of these three costs. In [27], the authors formulated an equation to evaluate the proposed efficient partitioning algorithm for distributed virtual environment, by considering the workload and communication cost. These are the same as our computation cost and communication cost. In [33], the authors considered the latency as another measure to evaluate the distributed systems as an addition to the computation and communication cost. In our case, migration cost is introduced in our evaluation framework. We treat the communication cost and migration cost differently, because they are different in two aspects. Firstly, migrating an agent between servers involves ownership transfer between partitions which may take longer than just transferring data between servers. Secondly, the receiver needs to process the ownership transfer which is different from processing the communication data. The difference between communication cost and migration cost is obvious by comparing the benchmark values.

In order to make accurate evaluation, the three metrics are measured through benchmark values. Benchmark values are collected by running tests, given the hardware and

software configuration (such as the CPU, network connection etc.) in the actual partitioned system to be implemented. The framework can be adapted to evaluate the performance of partitioning algorithms for synchronized distributed simulation on any distributed system, provided that the benchmark values are identified.

The following equations illustrate the detailed calculations used in the framework to evaluate the performance of the partitioning algorithm. The cost incurred on the server for partition i (referred to as server i hereafter) at simulation step j is:

$$\begin{aligned}
 cost(i, j) = & C_{\alpha}(i) * N_{\alpha}(i, j) \\
 & + C_{\beta}(i) * N_{\beta}(i, j) \\
 & + C_{\gamma}(i) * N_{\gamma}(i, j)
 \end{aligned}
 \tag{Eq. 3.1}$$

where

$C_{\alpha}(i)$: migration benchmark value of server i

$N_{\alpha}(i, j)$: number of migrations of server i at step j

$C_{\beta}(i)$: communication benchmark value of server i

$N_{\beta}(i, j)$: number of communications of server i at step j

$C_{\gamma}(i)$: computation benchmark value of server i

$N_{\gamma}(i, j)$: number of agents of server i at step j

$N_{\alpha}(i, j)$, $N_{\beta}(i, j)$, and $N_{\gamma}(i, j)$ are performance data which are collected during the simulation and generated for an analyzer of a particular partitioning algorithm for server i at step j .

Since the servers are synchronized at each simulation step, the maximum cost among the costs of the simulated servers is selected as the step cost. The cost incurred at simulation step j is:

$$cost(j) = \max cost(i, j) \text{ for all server } i \tag{Eq. 3.2}$$

The total cost of one partitioning algorithm is a summation of the costs incurred at each simulation step under that partitioning algorithm in a scenario. This is given by:

$$cost = \sum_{j=1}^S cost(j) \quad (\text{Eq. 3.3})$$

where S is the number of simulation steps. By comparing the step costs and the total cost, we can evaluate the performance of each partitioning algorithm for a given scenario.

These formulas are valid when the following two assumptions are not violated. Firstly, migration cost, communication cost and computation cost are not concurrent. This assumption that computation and communication are in separate phases is adopted from the bulk synchronous parallel model [9] and is used to simplify the analysis. This sets a lower bound on what kind of performance can be achieved using distributed processing. In real world execution, depending on the middleware and the implementation method, computation and communication can overlap and be carried out concurrently, which will give performance higher than the lower bound derived from our analysis. Secondly, the corresponding benchmark values of the three costs are fixed and do not change with the load of the server and with the traffic between servers. The second assumption is not violated when the server is not extremely overloaded and the communication link is not jammed. If such case does happen in the real distributed system, we need to model the computation benchmark value as a function of server load and the communication benchmark value and the migration benchmark value as a function of the load of the communication link. To simplify the evaluation process, in the following case study, we only consider the situation where servers are not extremely overloaded and there are sufficient bandwidth between servers.

3.4 Validation of the Framework and Performance Evaluation Process

We designed a real distributed system to validate the proposed framework and performance evaluation process by comparing the predicted partitioning performance with that of the real distributed simulation.

Table 3.1: Analysis Metrics

<i>Number of Messages</i>	The number of agents' states are retrieved from Server s_j to s_i . It is determined by the interest management
<i>CPU load</i>	We approximate the CPU load of each node over time by recording the total number of agents on each.
<i>Number of Migrations</i>	A migration happens when an agent a residing on server s_i at time t moves to server s_j , at time $t + 1$, $s_i \neq s_j$.

Three metrics are considered during analysis: *Number of Messages*, *CPU load*, and *Number of Migrations*(See Table 3.1). During the analysis, these metrics are collected for each partition. They are combined with some measured real costs to produce an overall measure of performance for the considered partitioning algorithm.

Experiments were performed to estimate real values for the various associated costs: computation, communication, and migration. The distributed simulation system to be developed will use desktop computers with the following hardware configuration: *Intel(R) Core(TM)2 CPU 6700@2.66GHz, with 2.00GB of RAM*. The computers are connected via *100MB Ethernet*. The distributed simulation will be *HLA/RTI* [10] compliant and implemented using the DMSO *RTI1.3NG-V6*. The communication cost and migration cost were estimated using a test case with two computers each running a federate. With the HLA/RTI there is an associated synchronization overhead at each time step due to *time advance requests*. To approximate this we configured a test case with two federates sending no messages and calculated the time needed to execute one hundred time advance

requests. To estimate the communication cost, a simulation with two federates was setup and one hundred messages were passed between them. The synchronization cost is factored out and then the execution time is divided by the total number of messages to approximate the cost of sending a single message. We adopt this strategy rather than a single round trip time to account for packet buffering which may occur in the underlying network infrastructure (we assume that with buffering sending one hundred messages will not be as expensive as one hundred round trip costs). The cost of a single migration is calculated in the same way, but rather than sending messages each machine performs a series of migrations. The ownership transfer between federates is also considered in the cost. The computation cost value is the average time that the sequential simulator takes to execute a single agent at one simulation step. The costs obtained from the experiments are shown in Table 3.2.

Table 3.2: Calculated Costs

Metric	Cost(ms)
Computation C_α	8.074
Communication C_β	6.02
Migration C_γ	51.01

We assume a simple agent architecture where agents do very little deliberation. However, the analyzer can easily be adapted to evaluate situations with much heavier weighted agents which have large working and state memory. The importance of the costs is not their absolute values but the relative weight among them. Table 3.2 shows that for this case, migration is 8.5 times more expensive than communication and a little over 6.3 times more expensive than computation. It is the relative cost of each metric which determines the effectiveness of a partitioning algorithm.

To validate the analyzer, we need to compare the results generated from the analyzer with those from real experiments. To do this we tested the two partitioning algorithms

mentioned above, which are FALB and static partitioning, in various configurations. The test case considered has 200 agents moving within a square environment of 100x100. Each of the agents is initialized with a random location within the environment and moves from left to right with a fixed velocity of $(0.5, 0)$. Once the x location of an agent reaches 100 it moves back to the center, $x = 50$. The simulation runs for 200 time steps with 2 and 4 nodes and the total execution time is then recorded. This test case essentially starts with an evenly balanced distribution of agents in the environment and then over time the right side of the environment becomes more heavily loaded.

The analyzer does not account for the inherent overhead caused by HLA/RTI synchronization. The real system is therefore first tested without any agents or communication to calculate the synchronization overhead. The synchronization overhead is then subtracted from the result obtained from the real system for direct comparison with the analyzer. This overhead exists in all cases and is only related to the number of federates, so it should not invalidate comparisons made among different partitioning algorithms using the analyzer.

Figure 3.3: Analyzer Prediction and Real System Output for Static Algorithm (2 Nodes)

Figures 3.3 and Figure 3.4 show the comparison between what the analyzer predicts and the real system output for static algorithm. It clearly shows that the analyzer's

Figure 3.4: Analyzer Prediction and Real System Output for Static Algorithm (4 Nodes)

Figure 3.5: Analyzer Prediction and Real System Output for FALB Algorithm (2 Nodes)

predictions are very close to the outputs of the real system. Figures 3.5 and Figures 3.6 show how the analyzer's predictions compare with the real execution results in the case of the FALB algorithm. Again the results are very similar. However, in all cases the analyzer's results are slightly less than those obtained from the real executions. This is to be expected as the analyzer does not consider the overhead associated with the execution of partitioning algorithms. The key issue, as the results show, is that the analyzer's predictions follow the overall trend of the real system for a variety of different parameters.

Figure 3.6: Analyzer Prediction and Real System Output for FALB Algorithm (4 Nodes)

3.5 Case Study

In this section, we present a case study to demonstrate the usefulness of the framework in predicting the performance of different partitioning algorithms for an agent-based simulation system without actually implementing the distributed system. The agent-based simulation system under study was developed to simulate a huge crowd for military operations. It is a sequential agent-based simulation system. In order to make it scalable for a large number of agents, a distributed simulation should be implemented. We applied the framework to help us select a suitable partitioning algorithm for different scenarios.

3.5.1 Partitioning algorithms

In our case study, the partitioning algorithms considered are the static partitioning algorithm and the FALB. Under the static partitioning algorithm, the configuration of each partition is predetermined in the simulation initialization phase and does not change during the simulation. While under the FALB algorithm, which is a dynamic partitioning algorithm, the configuration of each partition is initialized at the beginning of the simulation and changes during simulation in order to ensure that the loads among different

partitions are evenly distributed. Under both algorithms, communication cost is incurred when partitions communicate with each other.

For both the static and the FALB algorithms, the migration is triggered when an individual moves from one partition to another partition. However, for the FALB algorithm, when the loads among partitions are imbalanced, a repartition, which changes the configuration of each partition, will be triggered. The repartition may result in a large number of migrations among different partitions. Under the static partitioning algorithm, since the configuration of each partition does not change during simulation, repartition is not required.

3.5.2 Benchmark Values

In 3.4, we have described how the experiments were set up to estimate benchmark values for the various associated costs: communication, computation and migration. Since we used the same hardware and software configurations for validation and case study, the same set of benchmark values were used for the case study.

3.5.3 Scenario

To simplify the analysis, we define a scenario with agents in a 100*100 area under normal and emergency situation. There is only one exit in the middle of northern wall. The agents are randomly located at the initialization phase and wandering or flocking in a group during normal situation. In emergency situation, the agents will evacuate via the exit. In [28], a detailed behavior model is described. By studying two different situations, we aim to find a proper partitioning algorithm that can be adapted to the situations when the agents are either uniformly distributed or clustered together in the virtual environment.

Figure 3.7: Performance under Normal Situation

Figure 3.8: Performance under Emergency Situation

3.5.4 Performance Analysis

Figure 3.7 shows the performance of static partitioning algorithm and FALB algorithm under normal situation. From the figure, we can conclude that the two partitioning algo-

gorithms have almost identical performance under normal situation. The detailed step cost is not shown here, because they are almost identical for the two partitioning algorithms under normal situation. Either algorithm can be selected to implement the distributed agent-based system under normal situation.

Figure 3.9: Load Imbalance Factors among Partitions under Emergency Situation

Figures 3.8, 3.9 and 3.10 show the performance under emergency situation. From Figure 3.8, we can see that the total cost for the FALB algorithm is lower than that of static partitioning algorithm. However, under both partitioning algorithms, the total cost increases as the number of agents increases.

Figure 3.9 shows the load imbalance factors among partitions. The load imbalance factor is defined as the maximum load divided by the minimum load among partitions. Both the maximum and minimum load are in terms of the number of agents in a partition. The figure shows that the load under the FALB algorithm is better balanced among servers than that under the static partitioning algorithm. We can conclude that the

Figure 3.10: Step Cost under Emergency Situation

computation load is distributed evenly under the FALB algorithm; while the computation load is skewed under the static partitioning algorithm.

Figure 3.11: Partitions under Clustered Situation

Figure 3.10 shows the step cost under the FALB algorithm is less than that of static

partitioning algorithm until the last 10 simulation steps. In the late stage of the simulation under emergency situation, the agents are highly clustered at the exit. Although the load is well balanced under the FALB algorithm the step performance is almost the same as that of the static partitioning algorithm. It is because the communications between each partition are increasing dramatically in the clustered situation for both partitioning algorithms and the communication cost becomes a dominating component of the overall performance.

Figure 3.12: Communication Cost under Emergency Situation

Figure 3.11 shows the partitions at the late stage of the emergency situation when individuals (dots) are clustered at the exit under both algorithms. The square point at the bottom is a threat (e.g., explosion, or suicide bomber). And circles around it are its effective ranges. The thin black lines are the borders of the partitions under the static partitioning algorithm; while the thick black lines are the borders for the partitions under the FALB algorithm. It is obvious that there are a large number of individuals around the borders of partitions under both algorithms, which results in a large amount of communications between partitions.

Figure 3.13: Percentage of Communication Cost under Emergency Situation

Figure 3.12 shows the communication cost at each simulation step. It is obvious that the communication costs increase dramatically at the late stage of the simulation under emergency situation, especially for the FALB algorithm. The result is identical with our analysis of Figure 3.10. Figure 3.13 shows the ratio between the communication cost and the step cost. The figure further confirms that the communication cost becomes a dominating factor of the performance at the late stage of the simulation under the emergency situation.

In conclusion, by comparing both algorithms under emergency situation, the FALB algorithm is better in performance measure. But it may not be the perfect partitioning algorithm for the emergency situation especially under the clustered situation as shown in Figure 3.11. Other partitioning algorithm or modification of the fast adaptive load balancing algorithm, which can reduce the communication costs under highly clustered environment, should be investigated.

3.6 Summary

A generic framework was proposed to evaluate the performance of partitioning algorithms for an agent-based simulation system. A performance evaluation process that monitors the performance of each partitioning algorithm under consideration at each simulation time step was also developed. The overall performance of each partitioning algorithm is formulated by analyzing the step cost logged. Experiments were conducted to validate the proposed framework and evaluation process by comparing the performance of real partitioned simulation system with the predicated performance. The results show that the analyzer's predictions follow the overall trend of the real system for a variety of different parameters. Furthermore, case study was designed to demonstrate the process of evaluating the performance of different partitioning algorithms and to draw conclusions from the performance evaluation of each partitioning algorithm.

In conclusion, the framework and the evaluation process can aid us to predict the performance of partitioning algorithms for a specific agent-based simulation under different scenarios without actually implementing the distributed systems. Using the results from the prediction, we can then implement an appropriate partitioning algorithm for the specific agent-based simulation system.

Chapter 4

Cluster Based Partitioning for Agent-based Crowd Simulations

4.1 Introduction

For agent-based crowd simulation, agents have been shown to naturally form groups and flow patterns. Several agents may share the same goals and move together for a period of time during simulation. The long term interest (i.e., agents' goals) are an important factor for an effective partitioning algorithm. Clustering is a technique used to group similar objects or values together. In our study, we aim to apply the clustering techniques to partition the crowd simulation to achieve balanced computation load, and minimized communication overload and minimized migrations at the same time. Two cluster based algorithms, namely K-means partitioning and grid-based partitioning are proposed. Several experiments are designed to further investigate the strengths and weaknesses of these two partitioning algorithms under different scenario setups.

4.2 Cluster-based partitioning

Clustering is a technique used to group similar objects or values together. The notion of similarity can be defined in a number of ways. For agent-based crowd simulation the objective is to cluster agents such that those which interact most frequently are

grouped in the same cluster. In general communication occurs when the AOIs (defined by the sensor range) of distributed agents overlap. If we assume that agents interact with objects and other agents within their AOI then agents within close proximity should be clustered together. This simple approach is fine for a given snapshot of a simulation and the agents should be clustered according to their position. However, as agents are constantly moving, clustering based on position alone may not always be the best option in the long run.

Figure 4.1: Group Crossing Pattern

Consider for example two distinct groups of agents moving perpendicular to one another. The two groups will overlap at some point in the future. Figure 4.1 illustrates this common scenario in crowd simulation. The most appropriate approach when partitioning this scenario depends on a number of factors: the number of time steps the groups remain overlapped, the relative size of the groups, the movement of the agents after overlapping, etc. The partitioning algorithm can do one of two things. The first approach is to merge the crossing agents into a cluster for the period in which they remain in close proximity. Then if the agents split after the intersection the cluster will split into the original configuration. The problem with such an approach is the cost of migrating the agents from

one partition to another partition, the overhead of which is far higher than for sending messages. The second approach is to retain the original clusters in some way, incurring extra communication overhead for the period that the two clusters overlap. Again the effectiveness of either approach depends on the duration of overlap and other factors.

In general, however, unlike N-body problems [3] (where gravity or other attractive forces cause clustering), agents will not always remain in the same clusters. The behavior of the agents will depend on the internal state of the agent and its beliefs and goals. We believe it may be possible to cluster agents based on a combination of the agents position, velocity and internal state. The assumption is that agents close to each other with the same behaviors are likely to interact now and also in the future.

4.2.1 K-means Partitioning

To cluster agents into groups we use the K-means [29] clustering algorithm which groups objects into K clusters based on some attributes that form a vector space. K-means has been applied in a wide variety of disciplines including Sociology, Geography and Computer Science. We consider the agents' positions as attributes, which span over a 2D euclidean vector space (i.e., the environment), in the following. Given n agents, k clusters each with center c_i , and agent a_i at position p_i , the K-means approach tries to minimize the following term:

$$\sum_{j=1}^k \sum_{i=1}^n \|p_i - c_j\|^2$$

The K-means clusters are calculated using Floyd's [14] algorithm¹. The cluster centers are initialized to the position of a randomly chosen agent and each agent is then assigned to the cluster with the closest centroid. Once every agent has been assigned to a cluster, the centroids of each cluster are recalculated by determining the center of

¹There are parallel algorithms which can be used to calculate the clusters [15]

mass of all the agents in the cluster. The algorithm then calculates the change in the centroid of each cluster and determines when the clustering has converged and hence terminating the procedure. If the change in centroid is not below some given threshold, then the process repeats, re-calculating the centroids and then the clusters. The K-means algorithm is executed at pre-defined intervals, with more frequent calculations resulting in better tracking of the partitions but at extra cost. However, the number of iterations required at each re-calculation is determined by the deviance of the current centroids from the previous ones. Therefore, longer periods between calculations may result in more iterations per calculation.

4.2.2 Grid-based Partitioning

Figure 4.2: Simple Grid-based Partitioning Demonstration

Compared with other clustering methods, the grid-based clustering is known for its

fast processing time [24]. The algorithm also offers a great deal of control as the execution time is proportional to the defined grid size rather than the data objects. These methods use a single uniform grid mesh to partition the entire problem domain into cells. Statistics are collected regarding the data objects (in our case agents) located within a cell, which form a summary of the cells' contents. Clustering is then performed on the grid cells based on this statistical summary of each cell. Since the granularity of the grid is usually chosen such that there are far fewer cells than data objects, the processing speed for clustering can be significantly improved. However, the granularity of the grid cell, which affects the effectiveness and performance, should be defined sufficiently small for a specific scenario.

Before we further investigate the grid-based partitioning method, it is necessary to clarify some terminologies. Figure 4.2 shows a part of the virtual environment in a simulation. There are 9 cells. In each cell, there are number of agents. Agents with different goals are represented by different shapes. Each cell calculates and maintains statistics of the agents in the cell. In our study, the statistics include the common goal position of agents in the cell g_{com} , variance of the agents' goal positions g_{var} , and the number of agents n . g_{com} is defined as the goal position that is shared with most number of agents in the cell. And g_{var} is defined as:

$$g_{var} = \frac{n - n_{g_{com}}}{n}$$

where $n_{g_{com}}$ is the number of agents that share the same goal. In our study, g_{var} is considered as high if it is greater than 10%. Cells high goal position variances (for example the center cell) will not be grouped into clusters. For cells with low goal position variance, we will try to group them into clusters. As shown in Figure 4.2, cells in light grey are clustered into cluster 1 since these three cells contain agents with the same goal positions. Likewise, cells in dark grey are clustered into cluster 2. Agents in the

remaining cells, which are not clustered into any cluster, can join clusters if they have the same goal positions as these clusters and are located within the cluster's *bounding box* expanded by agent's sense range. Each cluster has a bounding box, which is defined as a minimum rectangle that can accommodate all the agents in the cluster. For example, agents B and C can join cluster 1; Likewise agents D and agent E can join cluster 2. After joining a cluster, agents will be treated as cluster members. Generally, a *cluster* is a collection of agents with similar attributes (goal position in our study). In contrast, an *individual* is an agent which does not belong to any cluster. Agents A, F, H, and G are individuals in this case. A *partition* is defined as a collection of one or more clusters and individuals which will execute together on a single node or server. The number of partitions is determined by the number of servers provided for distributed simulation.

While the grid-based clustering algorithm aids us to identify clusters with similar attributes and also individuals, the result does not derive a balanced distributed system directly in terms of load and communication. Furthermore, the number of clusters identified by this algorithm may not coincide with the number of servers available. We need to further distribute the identified clusters and individuals into partitions. A rule based algorithm is proposed to determine appropriate splitting/grouping of clusters and individuals. It takes the load thresholds and the cluster statistics into consideration. As mentioned above, the interest management algorithm is also critical for ensuring optimum performance of the distributed agent-based simulation. In our proposed approach, we apply the concept of bounding box to ensure an efficient distributed simulation system with low communications among partitions.

As previously mentioned, the grid-based clustering has several characteristics that make it appropriate for identifying clusters and individuals. However, using it for partitioning is not straightforward, considering the interest management and load balancing issues. In the remainder of this section, we will discuss our distributed grid-based

partitioning algorithm with considerations of load balancing, interest management and migrations. Furthermore, we will also discuss the limitations of our current design.

The grid-based partition algorithm executes in two phases, namely initialization phase and regular phase. The initialization phase is performed once at the beginning of each simulation to initialize partitions. The regular phase is performed at each time step in every partition independently. Figure 4.3 illustrates the steps at each phase. We will discuss those steps in detail as follows.

1) Evenly Distribute Virtual Environment among Partitions: To improve the efficiency of the clustering procedure and fully exploit the distributed simulation architecture, a distributed version of the grid-based clustering algorithm should be employed. In the initialisation phase, the virtual environment is evenly distributed among partitions so that each partition contains equal area of the virtual environment. Each partition will perform the grid-based clustering algorithm independently to identify clusters and individuals that are located on that partition.

2) Perform Grid-based Clustering Algorithm Independently: Our grid-based clustering algorithm for agent-based simulation is identical to that of the grid-based clustering method used in data mining. It consists of 4 main steps. Each partition performs the algorithm independently.

Firstly, a uniform grid is created within the environment in order to distribute agents to cells based on their positions. Each cell is assigned an index to represent the cell and its relative position in the grid. The granularity of grid cell should be defined properly to accommodate different performance and effectiveness requirements for different scenarios. A smaller cell size incurs a higher cost for the clustering procedure, however, a cell size which is too large will result in inappropriate clustering (i.e., the average statistics of all the agents within a cell will be meaningless as the variance will be too high).

Secondly, agents will be allocated to cells according to their positions (i.e., an agent will be placed in a cell if it is within the cell boundary). The complexity of the assignment

Figure 4.3: Steps of Grid-based Partitioning Algorithm

algorithm is $O(N_{partition})$, where $N_{partition}$ is the maximum number of agents in each partition.

Thirdly, the statistics of each cell is calculated from the collection of agents contained in the cell. As mentioned before, the statistics include the common goal position of the agents, the variance of agents' goal positions and the number of agents. All three statistics are calculated by traversing through the list of agents once. The common goal

```
Generate a grid structure Hashtable<Index,Cell> grid;
Assign the individuals to cells;
Calculate the statistics:  $g_{common}$ ,  $g_{var}$ ,  $n$ ;
for each cell  $c$ 
    if ( $c.g_{var} < 0.1$  AND  $c$  not empty AND  $c$  not traversed)
        initialize a Cluster  $cl$ ;
        set  $c$  to traversed;
        add cell  $c$  to cluster  $cl$ ;
        addNeighbor( $cl, c$ );
    else
        set  $c$  to traversed;
    endif
    if ( $cl.cells.size < 2$ )
         $cl = null$ ;
    endif
endfor

addNeighbor(Cluster  $cl$ , Cell  $c$ )
     $nbs = getNeighbors(c)$ ;
    for each neighbor  $nb$  in  $nbs$ ;
        if ( $nb$  not empty AND  $nb$  not traversed AND
             $nb.g_{var} < 0.1$  AND  $nb.g_{com} == c.g_{com}$ )
            set  $nb$  to traversed;
            add cell  $nb$  to cluster  $cl$ ;
            addNeighbor( $cl, nb$ );
        endif
    endfor
```

Figure 4.4: Pseudo-code of Grid-based Clustering Algorithm

position of the agents in a cell acts as a measure to cluster neighbor cells. For instance, having the same common goal positions indicates the two neighbor cells share the same interest in a period of time. So, they will be grouped in the same cluster. The variance of agents' goal positions determines whether agents in the cell should be treated as cluster members or individuals. For instance, high goal position variance indicates a considerable number of the agents in the cell have different goal positions. In such circumstances the agents should be treated as individuals rather than cluster members. The complexity of calculating the statistics is the same as the last step.

The final step is the process of clustering the similar cells recursively by traversing grid cells. Figure 4.4 illustrates this process. Given the grid structure, each cell is labeled with a specific ID which corresponds to its relative location from the top left cell in the grid. Beginning with the top left cell, the cell merging process attempts to cluster neighbor cells based on their common goals positions. Once a neighbor cell with the same common goal position is identified, the algorithm will include the neighbor cell into a cluster, mark the neighbor cell as traversed and traverse the neighbor's neighbor cells. The process terminates when no cell with the same common goal position exists for the current cell. For the remaining cells, if a cell is not clustered, we will perform the same cell merging process in a sequential order according to its cell ID. The complexity of the worst case of cell merging is $O(K * m)$, given the number of cells is m in each partition and the maximum number of neighbors of a cell is K . After cell merging, each cluster will contain agents which are both similar in locality and goal (or interest).

3) *Exchange Bounding Boxes and Statistics:* As defined previously, the bounding box is the minimum rectangle that can accommodate all the agents in the cluster. Each cluster has a bounding box. The configurations (minimum and maximum x and y coordinates) of the bounding boxes are broadcast among the partitions. The statistics of each cluster are broadcast as well in order to reform partitions.

4) *Reform Partitions*: If two clusters are located in different partitions with bounding boxes overlapping with each other and have the same common goal position, a merge of these two clusters will occur. The merged cluster will be assigned to the partition (amongst all the overlapping partitions) with the least agents. After merging of clusters, the load on each partition (i.e., the number of agents) may not be balanced. A load balancing strategy is required to assign clusters and individuals to partitions evenly. For an overloaded partition, if there is a large cluster with a load which is larger than the partition's load threshold (in our case study, the partition's load threshold is defined as 1.2 times of the average number of agents among all the partitions), we will break this cluster into several smaller clusters with load less than the threshold. After breaking down large clusters, we will calculate the total load of clusters in the partition by summing up the load of each cluster. If the total load of clusters is larger than the partition's load threshold, we will assign some furthest cluster(s) from the partition center to the partition with lowest load until the total load of clusters is lower than the partition's load threshold. If the partition remains overloaded after the above operations, we will move individuals to partitions which have groups or individuals with nearest distance to them.

5) *Execute Agents*: The design of agent differs for different simulation scenarios. In our study, the agent model for COSMOS project [2] is used. Agents sense the virtual world and nearby agents. They filter the perceived information and make decisions upon that information. They can also move around in the virtual environment. As a result, we update agents' positions, inner states, perceptions, and decisions periodically during simulation. The perceptions are based on the current states of the virtual environment and nearby agents. The partitions will communicate with each other to retrieve necessary states of agents and virtual environment for the execution of perceptions (see step 7). An interest management algorithm is required to perceive the virtual environment and nearby agents with low communications.

6) *Update Bounding Boxes and Statistics:* Cluster's bounding boxes and statistics are updated after updating agents' states, since clusters are collections of agents.

7) *Communicate with Other Partitions:* The communications among partitions involves exchanging states for the interest management and load balancing. In our research we adopt the bounding box concept to define the area of interest. The bounding boxes assist us to perform the interest management. Bounding box coordinates for clusters as well as the statistics of each cluster are broadcast among partitions at each time step. The exchange of agents' states between two partitions happens in two cases as shown in Figure 4.5 and Figure 4.6.

Figure 4.5: Interest Management between Individual and Clusters

Figure 4.6: Interest Management between Clusters

In our design, an individual has an expanded rectangle bounding box with the length and width equal to two times of its sensor range. We also expand clusters' bounding

boxes by an agent's sensor range. Objects in the expanded bounding box are the required objects to assist individuals or agents in a cluster to sense and make decisions. Figure 4.5 and Figure 4.6 show two cases that a partition need to request for agents' states from other partitions. In Figure 4.5, the individual from partition 1 has an extended bounding box which overlaps a cluster's bounding box in partition 2. The intersection area, which is the shadowed area, is calculated and a request is sent to partition 2 with the coordinates of the intersection area. Partition 2 will then send partition 1 agent updates in the intersection area if any. Likewise, in Figure 4.6, the cluster from partition 1 extends its bounding box. The extended bounding box overlaps with another cluster's bounding box in partition 2. The same procedure as describe above will be performed. That is, partition 1 will request partition 2 to send it state updates for the agents located in the intersection area.

8) *Merge and/or Break Clusters*: If clusters with the same common goal positions overlap with each other, a merge will be triggered. If these clusters are from the same partition, the merge will happen within that partition. New cluster will be formed with its bounding box and statistics updated and broadcast. If these clusters are from different partitions, the merge will be performed in such a way that only the partition with the least agents will request other partitions to transfer the agents in the related clusters. After that, a merge will be performed locally. A cluster can be broken into several smaller clusters and/or individuals. The variance of each cluster will be checked at each simulation step to track whether it remains valid to treat the cluster as a collection of agents with the same interest. Once the variance is too high, the grid-based clustering algorithm will be performed on that cluster to break it into several small clusters.

9) *Balance Load*: During the simulation, the distributed simulation may become imbalanced and a re-balancing will be triggered when necessary. If a partition is overloaded, which means the partition's load is 1.2 times higher than the average load among all the

partitions, it will attempt to shed load to nearby partitions with the furthest cluster and individuals. On the other hand if a partition is under-loaded, which means the partition's load is 0.8 lower than the average load among all the partitions. it will attempt to fetch the clusters and individuals from other partitions. Load balancing here follows the same procedure as described in *Reform Partition* step except it occurs in every simulation step.

After executing the last step, the algorithm will loop back to Step 5, the "Execution Agents" step in Figure 4.3, until the simulation stops.

4.3 Experiments

Simulating agent-based models on distributed architectures is an appealing method of executing faster (utilizing multiple CPUs) and larger (distributed memory) simulations. However, distribution increases the amount of communication necessary between the divided parts of the model. The basic principle of K-means and grid-based partitioning algorithms is to reduce communication overhead and the number of migrations in distributed simulation by organizing the agents onto appropriate machines. What defines appropriate depends on many factors, including the dynamism of the simulation, and the density and arrangement of agents.

This section describes the experiments for the evaluation of the *K-means clustering algorithm (KCA)* and the *Grid-based partitioning algorithm (GPA)*. These algorithms are specifically developed to support parallel and distributed execution of agent-based simulation. We also compare their performance with two typical approaches to partitioning: *division by state (DBS)* and *random division of agents (DOA)*. The DBS is a simple spatial decomposition of the environment into k areas, with each area A_i assigned to a single computation node N_i . As agents move from area A_i to area A_j , they are migrated (their entire state sent across the network) from node N_i to N_j . The DOA approach evenly

assigns the agents to the computation nodes, for a total of M agents each node will maintain a random selection of $\frac{M}{K}$ agents for the entire simulation. An agent interact with other agents when they are within its AOI. To minimize the interactions, it is important to assign agents near each other in one partition to maintain locality. However, since DOA randomly assign agents among partitions, a large number of agents near each other may assigned to different partitions. It may results a huge number of communications among partitions. Both methods can be seen as common practical approaches, where DBS minimizes communication at the cost of migration and DOA minimizes migration at the cost of communication.

4.3.1 Experiments Setup

The algorithms are evaluated using a series of engineered (or artificial) case studies. These engineered case studies are intended to serve two main purposes: firstly, they are designed to investigate the performance of algorithms in extreme cases and secondly they are intended to highlight particular strengths and weaknesses of each approach.

The number of partitions is fixed at four for all the cases studies in our research. That is, simulation will be executed on four computation nodes. Each case study is parameterized with the following control variables: *number of agents* (density), and *grouping degree* (clustering). The grouping degree is defined as a pair (g, m) , where $g \in [0, 1]$ indicates the percentage of the population in groups and $m \in [0, M)$ is the number of groups. The group has simple leader-follower behavior as defined in [41]. m group leaders are selected at random and then $(n \times g) - m$ agents are assigned in turn to leaders, where n is the total number of agents. The result is m approximately balanced groups. The entire population is then formed by $n \times (1 - g)$ *individuals*, m *leaders* and $(n \times g) - m$ *followers*.

All experiments are executed for 200 seconds at a time step of 0.5 second and results are an average over 10 runs. AOI (sense range) of each agent is fixed at eight meters. We vary each parameter as depicted in Table 4.1.

Parameter	Range
Number Agents n	[250,500,750,1000]
Group Degree (Group % g)	[0.0,0.2,0.5,0.8,1.0]
Group Degree (Group Num m)	[0,2,4,8]

Table 4.1: Parameter space

Our general hypothesis is that in environments which are highly dynamic and have little or no grouping, there is little that can be done in terms of intelligent clustering. In the worst case, agents are uniformly distributed across the entire environment and move (as individuals) rapidly throughout the environment. Under such circumstances the best that can be done is to use a straightforward approach (DBS or DOA) which simply attempts to balance execution load and disregards communication or migration overhead. However, with more grouping, less dynamism and non-uniform density, we expect that there will be sufficient patterns within the agent's movement so that the algorithms will be able to determine appropriate clusters.

4.3.2 Case Study 1 - Effect of Grouping Degree

The first case study considers a simple rectangular environment of 100x100. Individual agents and leader agents are given randomly selected goal locations, once a goal is reached, new goals are randomly assigned. Agents either move alone or as a group. The number and size of these groups is determined by the grouping degree parameter. The environment is initialized with n agents placed uniformly throughout the environment. The principle of this test is to understand the algorithm's performance in open environments with no directed motion. Essentially this should provide the worst case scenario,

where there are few patterns in the motion of agents, making it hard to identify persistent clusters of agents.

Figure 4.7: Partitioning Performance for Different Number of Agents

Figure 4.7 shows the performance of different partitioning methods with varied number of agents in the simulation. Both g and m are set to zero in this scenario and agents move randomly in the virtual environment. As shown in the figure, the GPA resembles the DOA. Initially the GPA generates 4 balanced partitions with agents which are near to each other residing on the same partition. The partitions remains unchanged during simulation since no cluster pattern is shown in this case study; while agents move across the virtual environment. As a result, agents near to each other may not be on the same partition during simulation. This explains why the GPA resembles the DOA. The figure also shows that the execution time increases significantly for both GPA and DOA as the number of agents increases. The KCA offers the best performance in this scenario. It maintains locality by clustering agents near each other into one partition. Its execution time increases linearly with the increase of number of agents. The DBS achieves similar performance as the KCA in this case study. Since the agents are move randomly and individually, a static partitioning of the environment is a natural option.

Table 4.2: The Execution Time for Different Partitioning Algorithms (Number of Agents $n = 1000$ & Group % $g = 1$ & Group Num $m = 8$)

	GPA	DBS	KCA	DOA
T_{Exe}	1.32E+06	1.54E+06	1.2+E06	9.04E+07
T_{DOA}/T_{Exe}	68.3	58.4	72.3	1

The DOA algorithm offers the worst performance when group patterns are shown in the simulation. For instance, Table 4.2 compares the execution time for these four partitioning algorithms in a simulation scenario with 1000 agents. All the agents are formed in 8 groups. The table shows that the execution time of the DOA algorithm is 68.3, 58.4 and 72.3 times longer than that of the GPA, the DBS and the KCA respectively. The execution time of the DOA is not in the same scale as those of other partitioning algorithms. As a result, it will not be compared with other partitioning algorithms in the rest of the experiments.

Figure 4.8 demonstrates the partitioning performance for different partitioning algorithms with varied group percentage. There are 1000 agents and 8 groups. The GPA provides the worst performance when the group percentage is small. However, the performance improves significantly as the group percentage increases. When more agents are in groups, the GPA is capable of clustering more agents into clusters and agents remain in these clusters during the simulation. In this way, the GPA achieves better locality and reduces migrations as the group percentage increases. The execution time of the KCA increases at first and decreases later on as the group percentage increases. The K-means algorithm is subject to the misclassification problem when there are outlier objects. The individuals are considered as outliers in agent-based crowd simulation. The KCA may divide a naturally formed group into different partitions when there are individuals around due to the misclassification. A communication overhead is incurred when a natural group is divided. The larger the group, the more communication overhead incurs when the group is divided. As the group percentage increases from 0.2 to

0.8 and the number of groups remains constant, each group becomes larger. That is why the performance of the KCA worsens as the group percentage increases from 0.2 to 0.8. However, as the group percentage equals 1, the misclassification becomes unlikely. Therefore, the KCA's performance improves. When $g = 1$, both the GPA and the KCA have better performance than the DBS.

Figure 4.8: Partitioning Performance for Different Group Percentage

Figure 4.9: Partitioning Performance for Different Number of Groups

We further investigate the effect of the number of groups on the GPA and the KCA. In this experiment, there are 1000 agents in the simulation and the group percentage is

1. Every agent belongs to a group with a group leader. Groups move randomly in the virtual environment. Figure 4.9 demonstrates the partitioning performance for the GPA, KCA and DBS algorithms when there are 2, 4 and 8 groups in the simulation. When the number of groups is 2, the GPA and the KCA have a worse performance than the static partitioning. In this case, two large groups will be divided into 4 clusters by the GPA and KCA algorithms. Dividing one group into two and assigning two groups to two different computational nodes will result in a large amount of communications at every simulation step. This affects the performance of the GPA and KCA. In this situation, although the static algorithm may results in imbalanced load most of the time during the simulation, the amount of communication is much less than those of the GPA and KCA. When the number of groups equals to 4 and 8, which is equal to or greater than the number of computation nodes, the GPA and KCA provide better performance than the static one. In these situations, we can fully utilize the clustering techniques to group agents into clusters and assign them to the corresponding computation nodes. When agents move in groups during simulation, the GPA and KCA can more or less maintain the distribution of the agents. Therefore, there are less migrations of agents.

4.3.3 Case Study 2 - Effect of Moving Pattern

In the last case study, groups and individuals move randomly in the virtual environment. Groups may move with or across each other. Both the GPA and KCA are shown to have similar performance. In order to further compare the GPA and KCA, this case study uses an open square virtual environment as well as a virtual environment with predefined routes.

A circular movement scenario is designed to investigate the difference between the GPA and KCA. There are total 1000 agents in the open square environment. As shown in Figure 4.10, two experiments are designed, namely sparse groups and close groups.

Figure 4.10: Circular Movement Scenario

Figure 4.11: Partitioning Performance for Circular Movement

For the first experiment, eight groups are evenly placed in a circle. While for the second experiment, eight groups are divided into two sets of groups. Each set has four groups. The groups in the same set are located near to each other but far from the groups in the other set. Figure 4.11 shows the partitioning performance for different partitioning methods under these two experiments. For the first experiment, the GPA and KCA have an identical performance. The execution time for the GPA and KCA is about two thirds that of static partitioning. These two algorithms take consideration of the the dynamic

movement of the groups and maintain natural groups on their respective computational nodes. Therefore, their migration costs are minimized in this scenario. However, the execution time of the KCA increases for the second experiment, while the execution time of the GPA remains. In this scenario, the KCA may misclassify agents. It may divide a naturally formed group into two partitions as shown in Figure 4.10, hence introducing more communications and migrations.

Figure 4.12: Virtual Environment with Corridor

We also developed experiments using a virtual environment with a narrow corridor where agents have directed motion (i.e., where all agents follow a similar path) to investigate the groups split and merge problem. With all agents moving along a regular path, one would expect that the clustering algorithms utilize this information to partition the agents. Figure 4.12 illustrates the layout of the environment for this experiment. The arrows represent the flow of agents. The narrow corridor at the center is used to study the merge-split problem previously mentioned. The size of narrow corridor is important

in evaluating the performance of different algorithms. The ranges of values for the length and width of the corridor are shown in Table 4.3.

Parameter	Range
length l m	[10, 16, 20]
width w m	[2, 4, 8]

Table 4.3: Corridor Parameters

The motion of the agents here is controlled and is indicated by the arrows in Figure 4.12. There are 1000 agents in the scenario. Agents are divided into four groups. Group leaders will lead the group to follow the indicated path from one side of the virtual environment to the other side. Four groups are divided into two sides. Two groups are initialized at the upper side; while other two groups are located at the lower side. All the four groups will move through the corridor to the other side of the virtual environment. If the corridor is narrow, we can expected that groups from upper and lower sides will overlap while they are moving along the corridor. In contrary, if the corridor is wide, the groups will move in an orderly manner without overlapping.

Figure 4.13: Partitioning Performance for Different Corridor Widths

Figure 4.13 shows the performance of different partitioning algorithms with different

corridor widths. The corridor length is fixed at 16m in this experiment. When the corridor width equals to 2m, groups overlap when they navigate through the corridor. The KCA provides better performance than the GPA in this case. While when the corridor width equals to 8m, groups navigate through the corridor without overlapping. The figure shows that the GPA offers better performance than the KCA. The KCA clusters agents based on their locations to the centroids. A centroid is calculated as the average location of the agents in the corresponding cluster. When two groups of agents are close to each other, it is possible that some agents in one group are near the centroid of another group. As a result, agents in a natural group could be divided into two clusters and thus introducing communication between clusters. In this situation, the GPA should provide a better performance than the KCA. While as for the GPA, it clusters agents based on their locations as well as future goals. If two groups of agents have different goals, they will remain in their respective clusters even when two groups overlap with each other. However, when two groups overlap, it may be better to repartition agents according to locations as that in the KCA to minimize the communication cost. In this situation, the KCA could be a better choice than the GPA.

The partitioning performance improves for all three partitioning methods with the increase of the corridor width. When the corridor is widened, the groups crossing the corridor becomes separate from each other. It reduces the number of communication for all the three partitioning algorithms. The GPA and KCA provide significant improvement over the DBS. They can adapt to the dynamic movement of the groups to maintain a balanced load among computation nodes. In contrast, the DBS cannot ensure a balanced partitioning.

Figure 4.14 shows the performance of different partitioning algorithms with different corridor lengths. The corridor width is fixed at 4m. As shown in the figure, the execution time of all the three partitioning algorithms increases as the length of the corridor

Figure 4.14: Partitioning Performance for Different Corridor Lengths

increases. As groups move across the corridor, communication incurs between different partitions because some agents of one partition may have their AOIs intersected with agents of other partitions. The longer the corridor, the more communication cost will incur. As the GPA maintains the partition of agents, communication happens along the horizontal boundary between clusters. While the KCA will adjust the allocation of agents. The communication between cluster happens along the vertical boundaries among newly formed cluster. For long and narrow corridor, the agents belonging to the same group spread in wider area. Therefore, the GPA incurs more communication overhead than that of the KCA. The KCA provides better performance than the GPA in this experiment.

4.4 Summary

We examined the problem of load balancing for distributed systems and discussed the importance of interest management for a good partitioning algorithms. We also proposed K-means and grid-based partitioning algorithms for agent-based crowd simulation. We

examined the effectiveness and performance of the K-means and grid-based partitioning algorithms. Based on the experimental results we can conclude that the K-means and grid-based clustering techniques are effective when obvious flow patterns are shown in the agent-based crowd simulation. We also purposely designed several experiments to investigate the differences between K-means and grid-based partitioning approaches. The K-means partitioning algorithm clusters agents based on their locations. It faces a misclassification problem, which becomes worse when there are individuals (outlier agents) in the simulation and when two naturally formed groups are near to each other. The grid-based partitioning algorithm clusters agents based on their locations as well as their long term interests (e.g., their goals). However, when clusters overlap with each other for a long period of time, it may provide worse performance than repartitioning agents according to their locations.

Chapter 5

Conclusions

5.1 Conclusions

Partitioning algorithms have been studied in many fields including distributed virtual environments as well as agent-based simulation systems. We reviewed the main concepts of the partitioning algorithm and did a comparative study of modern heuristics for solving the partitioning problem in distributed virtual environments. Given the fact that the problem of partitioning DVEs is similar to that of partitioning agent-based simulations (Both involve assigning avatars or agents to servers), we aimed to adopt the techniques used in the distributed virtual environment to the agent-based simulation. We also categorized the current partitioning algorithms into four classes based on how they maintain state and how they distribute the load amongst servers.

Since one of our research objectives is to evaluate partitioning algorithms for agent-based simulation systems, we also reviewed the related research work on performance modeling and prediction for parallel and distributed computing systems. There are three common main techniques for performance modeling and prediction: measurement, mathematical modeling and simulation. In our study, we followed a hybrid approach which combines these three techniques. Measurement is used when collecting the benchmark values. Mathematical model is used to get the total cost of each simulation step. And

at last we simulate the effect of partitioning algorithm in a non-distributed system to predict the performance.

A generic framework was proposed and developed to evaluate the performance of partitioning algorithms for agent-based simulation systems. A performance evaluation process that monitors the performance of each partitioning algorithm under consideration at each simulation time step was also developed. We also developed distributed simulation system using static partitioning algorithm and fast adaptive load balancing algorithm to validate the proposed framework and evaluation process. The results show that our framework and evaluation process are capable of providing relatively precise prediction. In addition, we also demonstrated the process of evaluating the performance of different partitioning algorithms and drew conclusions from the performance evaluation of each partitioning algorithm. In summary, the framework and the evaluation process can aid us to predict the performance of partitioning algorithms for a specific agent-based simulation under different scenarios without actually implementing the distributed systems.

The other objective of our research is to apply clustering techniques to partition crowd simulation, taking consideration of movement patterns and long-term interests of agents. In order to achieve this, we examined the problem of load balancing for distributed systems and discussed the importance of interest management for a good partitioning algorithm. We also proposed and developed K-means and grid-based partitioning algorithms for agent-based crowd simulation. Test cases were also set up to examine the effectiveness and performance of the algorithms. From the test results, we can conclude that the K-means and grid-based partitioning algorithms offer better performance when obvious flow patterns are shown in crowd simulation. We also further investigated the differences between K-means and grid-based partitioning algorithms. The K-means partitioning algorithm clusters agents based on their locations. It faces a misclassification

problem. The problem become worse when there are individuals (outlier agents) in the simulation and when two naturally formed groups are near to each other. The grid-based partitioning algorithm clusters agent based on their locations as well as their long term interests. However, when clusters overlap with each other, it may provide worse performance than repartitioning the overlapped clusters based on localities as what K-means does.

In conclusion, the research reported in this thesis achieved the two main objectives as stated in Chapter 1. A generic framework and evaluation progress were proposed and validated to evaluate the performance of partitioning algorithms for agent-based simulation. The K-means and Grid-based partitioning algorithms were developed to partitioning agent-based crowd simulation by utilizing the move patterns which are generally shown in crowd simulation.

5.2 Future Work

The K-means and grid-based partitioning algorithms are evaluated using our proposed analyzer tool. They are capable of providing effective partitioning of crowd simulation in specific situation. However, in real world, the behavior of a crowd can change dramatically when an external event (e.g., terrorist attack, fire alarm etc.) happens in public events or environment (e.g., National Day Parade, MRT station, etc.). As the behavior of the crowd changes, the move patterns may change. It will be beneficial that we can switch among different partitioning algorithms to address different move patterns in different situations. The proposed framework and evaluation progress can also be applied to evaluate the benefit of switching partitioning algorithms before implementing the feature on the real distributed simulation system.

References

- [1] Crowd Dynamics. <http://www.crowddynamics.com>, 2007.
- [2] CrOwd Simulation for Military OperationS. <http://cosmos.ntu.edu.sg/>, 2007.
- [3] Sverre J. Aarseth. *Gravitational N-body Simulations: Tools and Algorithms*. Cambridge University Press, 2003.
- [4] V. Adve, R. Bagrodia, J. Browne, E. Deelman, A. Dube, E. Houstis, J. Rice, R. Sakellariou, D. Sundaram-Stukel, P. Teller, and M. Vernon. POEMS-End to End Performance Models for Dynamic Parallel and Distributed Systems. pages 160–163, Feb 1999.
- [5] V. S. Adve, R. Bagrodia, J. C. Browne, E. Deelman, A. Dube, E. N. Houstis, J. R. Rice, R. Sakellariou, D. J. Sundaram-Stukel, P. J. Teller, and M. K. Vernon. Poems: End to End Performance Models for Dynamic Parallel and Distributed Systems. *Software Engineering, IEEE Transactions on*, 26(11):1027–1048, Nov 2000.
- [6] S. Ali and M. Shah. A Lagrangian Particle Dynamics Approach for Crowd Flow Segmentation and Stability Analysis. pages 1–6, June 2007.
- [7] E. R. Beeker and E. H. Page. A Case Study of the Development and Use of a MANA-Based Federation for Studying U.S. Border Operations. In *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, pages 841–847, Dec. 2006.

- [8] C. E. Bezerra, F. R. Cecin, and C. F. R. Geyer. A3: A Novel Interest Management Algorithm for Distributed Simulations of MMOGs. In *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium*, pages 35–42, Oct. 2008.
- [9] Rob H. Bisseling. *Parallel Scientific Computation: A Structured Approach Using BSP and MPI*. Oxford University Press, March 2004.
- [10] James O. Calvin and Richard Weatherly. An Introduction to the High Level Architecture (HLA) Runtime Infrastructure (RTI). <http://dss.ll.mit.edu/dss.web/96.14.103.RTI.Introduction.html>, 1996.
- [11] Yuefan Deng, Ronald F. Peierls, and Carlos Rivera. An Adaptive Load Balancing Method for Parallel Molecular Dynamics Simulations. *J. Comput. Phys.*, 161(1):250–263, 2000.
- [12] M. Dorigo and L. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. *IEEE Trans. Evolut. Comput.*, 1997.
- [13] A. Ferscha and J. Johnson. *Performance Oriented Development of SPMD Programs Based on Task Structure Specifications*, volume 854/1994 of *Parallel Processing: CONPAR 94 VAPP VI, Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Sept 6-8 1994.
- [14] Floyd and W. Robert. Algorithm 97: Shortest Path. *Commun. ACM*, 5(6):345, 1962.
- [15] Attila Gürsoy. Data Decomposition for Parallel K-means Clustering. In *PPAM*, pages 241–248, 2003.

- [16] Anthony Guye-Vuillieme and Daniel Thalmann. A High Level Architecture for Believable Social Agents. *Virtual Reality Journal*, 5:95–106, 2001.
- [17] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, California, 2001.
- [18] Randy L. Haupt and Sue Ellen Haupt. Practical Genetic Algorithms. *Ed. Willey*, 1997.
- [19] M. Hribersêk and G. Kuhn. Domain Decomposition Methods for Conjugate Heat Transfer Computation by BEM. *Fifth World Congress on Computational Mechanics, Vienna, Australia*, 2002.
- [20] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings. Predictive Performance and Scalability Modeling of a Large-Scale Application. pages 39–39, Nov 2001.
- [21] David E. Keyes. Domain Decomposition in the Mainstream of Computational Science. *Proceedings of the 14th International Conference on Domain Decomposition Methods in Cocoyoc, Mexico*, 2002.
- [22] Michael Lees, Brian Logan, and John King. HLA Simulation of Agent-Based Bacterial Models. In *Proceedings of the 2007 European Simulation Interoperability Workshop*, Genoa, June 2007. Simulation Interoperability Standards Organisation. 07E-SIW-032.
- [23] Michael Lees, Brian Logan, and Georgios Theodoropoulos. Distributed Simulation of Agent-based Systems with HLA. *ACM Transactions on Modeling and Computer Simulation*, 17(3):11, 2007.

- [24] W. K. Liao, Y. Liu, and A. Choudhary. A Grid-based Clustering Algorithm Using Adaptive Mesh Refinement. *Lake Buena Vista, Florida, USA: 7th Workshop on Mining Scientific and Engineering Datasets in conjunction with SIAM International Conference on Data Mining (SDM)*, page 61C69, April 2004.
- [25] Chu Cheow Lim, Yoke Hean Low, and Wentong Cai. A Parallelism Analyzer Algorithm for a Conservative Super-step Simulation Protocol. In *Hawaii International Conference on System Sciences (HICSS-32), Maui*, pages 5–8, 1999.
- [26] Chu Cheow Lim, Yoke Hean Low, Boon Ping Gan, Sanjay Jain, Wentong Cai, Wen Jing Hsu, and Shell Ying Huang. Performance Prediction Tools for Parallel Discrete-event Simulation. In *PADS '99: Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 148–155, Washington, DC, USA, 1999. IEEE Computer Society.
- [27] C. S. Lui and M. F. Chan. An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems. *IEEE Trans. Parallel and Distributed Systems*, 13, 2002.
- [28] Linbo Luo, Suiping Zhou, Wentong Cai, Malcolm Yoke Hean Low, Feng Tian, Yongwei Wang, Xian Xiao, and Dan Chen. Agent-based Human Behavior Modeling for Crowd Simulation. *Comput. Animat. Virtual Worlds*, 19(3-4):271–281, 2008.
- [29] Youssef M. Marzouk and Ahmed F. Ghoniem. K-means Clustering for Optimal Partitioning and Dynamic Load Balancing of Parallel Hierarchical N-body Simulations. *J. Comput. Phys.*, 207(2):493–528, 2005.
- [30] R. Minson and G. Theodoropoulos. An Adaptive Interest Management Scheme for Distributed Virtual Environments. In *Principles of Advanced and Distributed Simulation, 2005. PADS 2005. Workshop*, pages 273–281, June 2005.

- [31] Graham Morgan, Fengyun Lu, and Kier Storey. Interest Management Middleware for Metworked Games. In *I3D '05: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, pages 57–64, New York, NY, USA, 2005. ACM.
- [32] P. Morillo, M. Fernandez, and J. M. Orduna. A Comparison Study of Modern Heuristics for Solving the Partitioning Problem in Distributed Virtual Environment systems. *Computational Science and Its Applications ICCSA 2003*, 2669/2003:985, Jan 2003.
- [33] P. Morillo, S. Rueda, J. M. Orduna, and J. Duato. A Latency-Aware Partitioning Method for Distributed Virtual Environment Systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(9):1215–1226, 2007.
- [34] Katherine L. Morse. Interest Management in Large-Scale Distributed Simulations. Technical report 96-27, Department of Information and Computer Science, University of California, Irvine., 1996.
- [35] Katherine L. Morse, Lubomir Bic, and Michael Dillencourt. Interest Management in Large-Scale Virtual Environments. *Presence: Teleoper. Virtual Environ.*, 9(1):52–68, 2000.
- [36] Ton Oguara, Dan Chen, Georgios Theodoropoulos, Brian Logan, and Michael Lees. An Adaptive Load Management Mechanism for Distributed Simulation of Multi-Agent Systems. In *Proceedings of the Ninth IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2005)*, pages 179–186, Montreal, Quebec, Canada, October 2005. IEEE Press.
- [37] Sungju Park, Dongman Lee, Mingyu Lim, and Chansu Yu. Scalable Data Management Using User-based Caching and Prefetching in Distributed Virtual Environments. In *Proceedings of the ACM symposium on Virtual reality software and technology, VRST '01*, pages 121–126, New York, NY, USA, 2001. ACM.

- [38] Mikel D. Petty, Frederic D. McKenzie, Ryland C. Gaskins, and Eric W. Weisel. Developing a Crowd Federate for Military Simulation. *Proceedings of the Spring 2004 Simulation Interoperability Workshop (SIW)*, pages 483–493, 2004.
- [39] S. Pllana and T. Fahringer. Performance Prophet: A Performance Modeling and Prediction Tool for Parallel and Distributed Programs. pages 509–516, June 2005.
- [40] Mauricio G. C Resende and Celso C. Ribeiro. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 2001.
- [41] C. W. Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model. In *SIGGRAPH '87 Conference Proceedings*, pages 25–34, 1987.
- [42] Pllana Sabri, Brandic Ivona, and Benkner Siegfried. Performance Modeling and Prediction of Parallel and Distributed Computing Systems: A Survey of the State of the Art. pages 279–284, April 2007.
- [43] Bernd Schattenberg and Adelinde M. Uhrmacher. Planning Agents in James. In *Proceedings of the IEEE*, volume 89, page 158C173, 2001.
- [44] H. Schwetman. Model-based Systems Analysis Using CSIM18. volume 1, pages 309–313 vol.1, Dec 1998.
- [45] Wei Shao and Demetri Terzopoulos. Autonomous Pedestrians. *Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005)*, 2005.
- [46] Barry G. Silverman, Gnana Bharathy, Kevin O’Brien, and Jason Cornwell. Human Behavior Models for Agents in Simulators and Games: Part II: Gamebot Engineering with PMFserv. *Presence*, 15(2):163–185, 2006.

- [47] Barry G. Silverman, Michael Johns, Jason Cornwell, and Kevin O'Brien. Human Behavior Models for Agents in Simulators and Games: Part I: Enabling Science with PMFserv. *Presence*, 15(2):139–162, 2006.
- [48] S. Singhal and M. Zyda. Networked Virtual Environments. *ACM Press*, 1999.
- [49] Mankyu Sung, Michael Gleicher, and Stephen Chenney. Scalable Behaviors for Crowd Simulation. *Computer Graphics Forum*, 23(3):519–528, 2004.
- [50] G. Vigueras, M. Lozanoand, J. M. Ordu, and F. Grimaldo. A Comparative Study of Partitioning Methods for Crowd Simulations. *Applied Software Computing*, 10(1):225–235, 2010.
- [51] Jinsheng Xu and Moon Jung Chung. Predicting the Performance of Synchronous Discrete Event Simulation. *IEEE Transactions on Parallel and Distributed Systems*, 15(12):1130–1137, Dec 2004.
- [52] M. Marzouk Youssef and F. Ghoniem Ahmed. K-means Clustering for Optimal Partitioning and Dynamic Load Balancing of Parallel Hierarchical N-body Simulations. *J. Comput. Phys.*, 207(2):493–528, 2005.