*Proceedings of the 2009 Winter Simulation Conference*
*M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, eds.*

# CLUSTER BASED PARTITIONING FOR AGENT-BASED CROWD SIMULATIONS

Yongwei Wang
Michael Lees
Wentong Cai
Suiping Zhou
Malcolm Yoke Hean Low

Parallel and Distributed Computing Centre,
School of Computer Engineering,
Nanyang Technological University, Singapore 639798

## 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. Distributed simulation, however, introduces its own challenges, in particular how to efficiently *partition* the load between a number of machines. In this paper we introduce a method of partitioning agents onto machines using an adapted k-means clustering algorithm. We present, validate and use an analysis tool to compare the proposed clustered partitioning approach with a series of existing methods.

## 1 INTRODUCTION

Crowds are highly dynamic complex systems that have recently drawn significant research efforts from a variety of fields. Agent-based simulation has been the tool of choice for many researchers as it allows modelers to incorporate complex individual models of human behaviors and motion. Simulation studies of crowds have been applied in a variety of scenarios including emergency or evacuation situations (Santos and Aguirre 2004). While agent-based models afford higher complexity and finer granularity they also require more computational resource. Modelers are currently faced with a trade-off between the complexity of individuals in the crowd and the size of the crowd. This means for large crowds, modeling individuals with complex rules becomes infeasible, typically individuals will instead be modeled as simple particles. Distributed simulation techniques can be used as a solution to this problem, allowing large scale high granularity simulations. In distributed simulation a collections of machines (or nodes) are used to model various parts of the simulation. The simulation is then divided up with the set of agents and/or environment being shared equally amongst the machines. Distribution has been applied to agent-based crowd simulations previously (Lozano et al. 2007) with promising results. However, distribution introduces its own challenges. The machines must communicate and synchronize, ensuring each agent does not perceive contradictions in the environment. It may not be necessary for every agent to have a completely consistent view of the world but it is essential that it does so for any information it uses when making decisions. We adopt terminology from the field of Distributed Virtual Environments (DVEs) where this problem is termed *awareness*. When the level of distribution becomes large the cost associated with this communication can become significant which in turn reduces the overall performance of the system. With the objective of distribution being to speed-up the simulation, the overhead of communication must not outweigh any benefit achieved through distribution. Therefore, keeping communication overhead low is essential for improving the overall performance and scalability of the system.

The key problem in minimizing communication overhead is how to partition the state and agents evenly amongst the nodes while maintaining a low communication overhead. To do so it is necessary to partition the agents so that agents reside on the same nodes as the other agents with which they interact most. Crowd simulations are highly dynamic and agents will interact with different agents and different regions of the environment state at different times. Therefore, the ideal partitioning will change over time and the system may need to adapt or load balance to reflect the new optimal partitioning.

978-1-4244-5771-7/09/$26.00 ©2009 IEEE 

Tracking this ideal partitioning incurs additional overhead, another complication to consider when devising approaches to perform the dynamic load balancing.

Crowd formation and characteristics have been studied for patterns and predictable behavior using a variety of techniques including Lagrangian motion (Ali and Shah 2007). Individuals in crowds rarely act in random or unpredictable ways and it is often possible to see patterns or clusters forming in the crowd. We propose that it is possible to exploit this clustering and these patterns to develop a partitioning algorithm which should be highly effective in a large number of different crowd scenarios. In this paper we present an adapted k-means partitioning algorithm and compare this with two other partitioning approaches: fast adaptive (Zhang et al. 2009) and static environment partitioning. We show that for agent-based crowd simulation a partitioning approach which identifies the inherent clustering and patterns of crowds outperforms traditional methods.

The remainder of this paper is organized as follows, in the following section we present a selection of related work in the areas of partitioning for distributed agent simulation and pattern based partitioning. Section 3 introduces the concepts of agent-based crowd simulation and the COSMOS framework being developed at Nanyang Technological University (NTU) (Luo et al. 2008). The subsequent section introduces the notion of partitioning and outlines various types of approaches. Section 5 outlines the k-means based algorithm we propose for agent-based crowd simulation. The following sections describe our experimental work and conclusions in testing the cluster based approach.

## 2 RELATED WORK

In this section we look at previous work in the area of clustered partitioning and of partitioning in agent-based simulation and DVEs. To the best of our knowledge clustered partitioning has not been applied in distributed agent-based simulation. K-means has actually been applied to N-body physics problems as a means of partitioning (Marzouk and Ghoniem 2005) for parallel simulation. The paper investigates Vortex particle methods for fluid dynamics. The test case considered involves evaluation of vortical velocities in a transverse jet. The authors adopt a slightly modified algorithm which scales each of the clusters by some constant. This constant is then modified at runtime and is adapted according to the load imbalance among the various processors. While k-means will give a balanced partitioning in terms of number of entities it does not guarantee a balanced load in terms of CPU. This weighting factor enables the algorithm to skew the clusters according to the CPU load imbalance during the last calculation period. For their test cases they consider large numbers of particles (157297) using 1, 16, 64, 128, 512 and 1024 clusters and nodes. In the best case scenarios they report a parallel efficiency of 98% (overhead of 2%) with 1024 nodes. They also show that with dynamic load balancing a much narrower variation in load among the various processors is observed when the k-means approach is adopted. Finally and most importantly, the dynamic load balancing is shown to significantly reduce the overall calculation time required for the problem.

In (Oguara et al. 2005) an approach is presented which considers agents access patterns to the shared state as a means of partitioning agents onto processors. The framework uses a different model of shared state, with the common state of the environment residing on different processors to the agents. To reduce communication load the state is moved closer to 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 state 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 in a few time steps.

Many distributed multi-agent systems adopt a static environment decomposition (for example (Lees et al. 2007, Lees et al. 2007)). 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 adopted for simplicity and not because of some significant observation. The suitability of a given partitioning algorithm depends on many characteristics of an agent simulation, the density of agents, movement of the agents, distribution of agents, etc. A general approach which identifies clustering of the agents should provide a good solution in those cases where the environment is not fully packed with agents.

The problem of partitioning DVEs is very similar to partitioning agent-based simulations. Generally there are characters moving and interacting within an environment and the problem involves assigning avatars (clients) to servers and in doing so maintaining low latency between avatars without overloading any given server. In (Morillo et al. 2004) an analysis of various approaches to the problem of partitioning in DVEs are presented. An algorithm which uses an *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 pheromones that ant colonies use for locating food. Each ant intermittently places the pheromone along the path which they follow. Subsequent ants will then follow paths with high pheromone concentrations. The algorithm defines border avatars

(clients) 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 border avatars, see (Morillo et al. 2004) for details.

## 3 CROWD SIMULATION

Modeling crowds is a difficult but valuable problem which has been investigated from a number of research fields including: military simulation, safety engineering, architectural design and digital entertainment. Generating accurate and believable crowds remains an open problem due to its multi-faceted nature. Various behavioral models have been developed (Musse and Thalmann 2001, Sung et al. 2004); simulation architectures designed (Nguyen et al. 2005, Pan et al. 2005); virtual environment representations studied (Paris et al. 2006, Pettre et al. 2006). In addition, commercial crowd simulation middlewares, such as AI.Implant (Engenuity Technologies Inc 2006) and Massive (Massive Software 2007), have been successfully utilized in digital production applications (e.g., Lord of Rings, Age of Empires 3, and Hannibal v Rome).

Crowd simulation is studied at a variety of different levels using a number of different approaches and techniques. There are three core strategies in the literature: flow-based models, particle models and agent-based models. These methodologies offer different benefits for modeling crowds. With flow-based models for example, the resolution of the phenomena which can be investigated is sacrificed for scalability. These models can simulate large crowds and offer basic insight about the crowd as a whole. However, they will often neglect individual behaviors and assume the crowd inhabitants are generally homogeneous. At the other extreme agent-based models (individual models) have a complete model of each individual in the crowd. Every individual has different behaviors, emotions, and movement rules. The computational requirements of such high resolution modeling are significant and executing simulations with large numbers of these agents can be challenging. One key issue which is still an open problem in crowd simulation is choosing the correct level of abstraction for the model. The level at which a crowd is modeled determines the phenomena which can be studied and the questions which can be asked. The modeler must therefore know the significance of different aspects of the model and ensure relevant issues are included in the model. Again the trade-off of resolution and scale is important. Choosing the wrong scale can mean essential phenomena are not captured. Developing a scalable high resolution simulation should help overcome this problem. Implementing a distributed simulation of an agent-based model is one way to achieve this.

The COSMOS project (Luo et al. 2008) is investigating large scale ($\approx$ 100000 individuals), high resolution agent-based models of crowds. Each agent contains a complex model of human behaviors and different types of individuals have unique experience sets resulting in highly believable and accurate simulations. Route and Motion planning is done through a combined $A^*$ planner (Dechter and Pearl 1985) and RVO (van den Berg et al. 2008) collision avoidance mechanism which results in realistic emergent crowd motion. With such complexity and scale in the COSMOS framework it is necessary to execute the models using distributed simulation techniques. The key challenge when distributing these models effectively is that of partitioning and how agents should be assigned to processors.

## 4 PARTITIONING

The problem we investigate is how to dynamically determine a balanced assignment of agents to nodes while ensuring communication between nodes is kept to a minimum. For purposes of explanation we first define a few concepts. We assume each agent operates with an *Area of Interest* (AOI) (Area of interest is a term from DVEs, an equivalent term Sphere of Influence is defined for agent simulation (Logan and Theodoropoulos 2001)), which defines an area of the environment with which the character can interact (i.e., read and write to state). Messages between nodes are necessary when the AOI of an agent $A_i$, which belongs to a node $N_j$, includes an object or a different agent which does not belong to $N_j$. We can now state that in order to minimize communication cost over a period of time the agents should be partitioned such that overlaps of AOIs between agents on different servers are kept to a minimum. Obviously as the environment is dynamic the optimal partitioning changes over time and tracking this optimal is the key challenge of load balancing. Load balancing itself has associated costs. Firstly data must be collected and analyzed to determine the quality of the current partition and determine the need for re-partitioning. Secondly, to re-partition and re-balance load, agents must be migrated with their state from one node to another.

It is possible to classify partitioning approaches according to how they maintain state and how they distribute the load amongst processors. There are four main approaches: centralized virtual environment, replicated virtual environment, divided virtual environment with agents and divide the virtual environment independently. Using a centralized virtual environment involves a single machine maintaining the agents environment, agents are then distributed on the remaining machines. Agents all interact with the same single view of the world which means maintaining consistency is straightforward. However, when the environment becomes large and the number of agents becomes large the central server will become a bottle-neck. A

replicated approach involves executing the agents of different servers with each server also maintaining a complete copy of the entire virtual environment. This approach avoids the problem of a single communication bottleneck, however, maintaining consistency among the copies of the virtual environments becomes costly when the environment is highly dynamic and there are large numbers of machines. The third approach involves dividing the environment among the nodes. The set of agents modeled by each node then changes depending on which agents are within the region of the environment being modeled. In this approach techniques can be used to avoid the need to broadcast updates when maintaining consistency. It is not necessary for every node to have a completely consistent view of the world but it must have a consistent view of any state with which it interacts. The final approach is to decouple the division of state and agents completely, with two groups of nodes modeling the state and agents separately. Consistency is then maintained among the environment nodes only, but the number of machines required for the same distributed performance is higher (this is assuming the majority of the computational requirements are due to processing the agents).

## 5 CLUSTERED PARTITIONING

Crowds have been shown to naturally form groups and flow patterns (Ali and Shah 2007). Exploiting this with an appropriate partitioning algorithm should enable efficient distribution of crowd simulation. In this paper we investigate the application of a clustering based partitioning algorithm for distributed simulation of crowds. 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 simulation the objective is to cluster agents such that those which interact most frequently are grouped in the same cluster. For crowd simulation and agent simulation 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(In some cases they may wait for short periods) moving, clustering on position alone may not always be the best option in the long run. Consider for example two distinct groups of agents moving perpendicular to one another which will overlap at some point in the future.

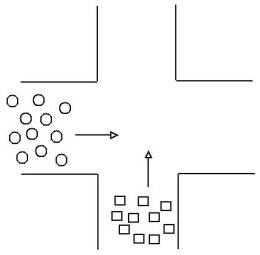

Figure 1: Group crossing pattern

Figure 1 illustrates this common scenario in crowd simulation with two groups merging at an intersection. The most appropriate approach when partitioning this scenario depends on a number of factors: the number of time steps the groups spend overlapping, the relative size of the groups, the movement of the agents after merging, etc. The partitioning algorithm can do one of two things. Firstly it can 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 node to node, the overhead of which is far higher than for sending messages. The second approach is to retain the original clusters in some way, inuring extra communication overhead for the period that the two clusters overlap. Again the effectiveness of either approach depends on the length of time of overlap and other factors. In general however, unlike N-body problems (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 being that agents close to each other with the same behaviors are likely to interact now and also in the future. In this paper however, we investigate the first scenario and attempt to re-cluster agents whenever their AOIs overlap.

To achieve the clustering of agents into groups we use the K-means (Marzouk and Ghoniem 2005) clustering algorithm which groups objects 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 which use statistics including Sociology, Geography and Computer Science. In this

paper we investigate a single method of clustering using the k-menas metric. We consider the agents positions as attributes which span a 2D euclidean vector space, the environment.

The method minimizes J given n agents; k clusters each with center $c_i$; and agent $a_i$ at position $p_i$:

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

For the analysis here the k-means clusters are calculated using Floyd's (Floyd 1962) algorithm. However, there are parallel algorithms which can be used to calculate the clusters (Gürsoy 2003). 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 each agent has been assigned to a cluster the centroids of each cluster are recalculated by determining the center of mass of all the agents in the cluster. The algorithm then calculates the change in the centroid of each cluster which it uses to determine when the clustering has converged and hence terminate 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. Therefore, longer periods between calculations may result in more iterations per calculation.

## 6 EXPERIMENTS

The experiments in this paper investigate the application of k-means partitioning to crowd simulation using an analysis tool described in previous work (Wang et al. 2009). Designing, developing and implementing new partitioning algorithms involves significant work. Different approaches may assume different underlying infrastructure and implementing the code necessary for each new algorithm can be a considerable effort. The analyzer tool was developed to solve this problem and allow rapid prototyping and testing of many partitioning algorithms. With the analyzer running in serial there is no need to implement complex message passing systems. Real costs for communication, computation and migration can be measured or varied between experiments. This provides a mechanism for evaluating each algorithm under different hardware configurations simply by altering these costs. To further motivate the use of the analysis tool and to validate its results we present an experimental validation, comparing results from a real system with that predicted by the analyzer.

### 6.1 Algorithms

We compare the clustered based partitioning algorithm outlined in section 5 with two other partitioning approaches: a grid based method and a static environment partitioning.

#### 6.1.1 Fast Adaptive Load Balancing

The Fast Adaptive Load Balancing (FALB) (Zhang, Jiang, and Li 2009) algorithm is an exension and form of grid generation load balancing methods (Deng et al. 2000). These approaches involve dividing the simulation space (environment) into cells arranged in a rectangular grid. Each cell is then assigned to a node and all agents contained within the cell are executed on that node. Load balancing is traditionally performed by moving the vertices of the cells such that heavily loaded cells effectively shrink, maintaining a smaller portion of the environment. In this paper we use the FALB method which adapts the edges of cells rather than vertices to re-adjust load. This approach has the advantage that the cells remain rectangular and thus the overhead associated with readjustment is reduced. If the vertices are free to move independently there can be a large overhead associated with adjusting vertices when the load is highly imbalanced. The scale by which the Line $L_i$ is adjusted is determined by the difference in load between the two cells which the line divides. An aggressiveness factor, $\varepsilon$, is used to determine how much cell scaling is done for a given difference in load.

#### 6.1.2 Static Partition

The final approach we compare with is a standard static division of the environment. The environment is divided into $n$ regions or cells and all agents within that cell are modeled by a single node. As agents move between cells they are migrated between the corresponding nodes. Clearly in some scenarios this method of decomposition will result in imbalance, especially when the crowd is highly clustered or skewed. The method should perform reasonably well in crowds of uniform density.

Table 1: Notation of costs

| | |
|---|---|
| $C_\alpha(i)$ | Cost of single migration form server $i$ |
| $N_\alpha(i,j)$ | Number of migrations of server $i$ at step $j$ |
| $C_\beta(i)$ | Cost of single message from server $i$ |
| $N_\beta(i,j)$ | Number of messages of server $i$ at step $j$ |
| $C_\gamma(i)$ | Cost to execute one agent on server $i$ |
| $N_\gamma(i,j)$ | Number of agents of server $i$ at step $j$ |

Using this method it is possible to decompose the environment in a number of ways: in a regular grid, in vertical/horizontal strips or in an irregular grid. In these experiments we consider a regular grid division. This seems the most appropriate choice as it minimizes surface area to volume ratio of each cluster, which should in turn minimize communication.

### 6.2 Analyzer Validation

To evaluate the partitioning algorithms we use an analysis tool developed to analyze output traces from the COSMOS crowd simulations (Wang et al. 2009). The analyzer can simulate and evaluate various partitioning algorithms, the assumptions and characteristics about the underlying hardware can be specified for each experiment also. The system attempts to partition the agents according to the selected algorithm and measures communication, load and migration costs for each. Three metrics are considered during analysis:

- *Number of Messages* - An agent $a$ residing on server $s_1$ requires a message whenever its sensor range overlaps with an agent $b$ on a server $s_2$, $s_1 \neq s_2$. (Note we assume the only dynamic state here is the agents position)
- *CPU load* – We approximate the CPU load of each node over time by recording the total number of agents on each.
- *Number of Migrations* – A migration happens when an agent $a$ residing on server $s_1$ at time $t_1$ moves to server $s_2$, at time $t_2$, $s_1 \neq s_2$.

During the analysis, the above metrics are collected for each partition. These metrics are combined with some measured real costs to produce an overall measure of effectiveness for the considered partitioning mechanism. The following equations detail the calculations used in the analysis tool to evaluate the performance. We assume a time-step simulation with all servers synchronizing and blocking at the end of every time step. To calculate the overall cost of a partitioning algorithm we analyze the cost at each time step. The cost incurred on the server for partition $i$ (referred to as server $i$ hereafter) at simulation step $j$ is:

$$\begin{aligned} cost(i,j) &= C_\alpha(i) * N_\alpha(i,j) \\ &\quad + C_\beta(i) * N_\beta(i,j) \\ &\quad + C_\gamma(i) * N_\gamma(i,j) \end{aligned} \tag{1}$$

$N_\alpha(i,j)$, $N_\beta(i,j)$, and $N_\gamma(i,j)$ are performance data which are collected during the simulation and generated from an analysis of a particular partitioning mechanism for server $i$ at step $j$. The overall performance criteria of any partitioning algorithm is simulation execution time, which is determined by the slowest server. We define the step cost of a partitioning algorithm (overall cost per time step) as the maximum cost of any single partition at each time step. The step cost of a partitioning algorithm incurred at simulation step $j$ is then:

$$cost(j) = \max(cost(i,j)) \ \forall \text{ server } i \tag{2}$$

The total cost of one partitioning mechanism for a given scenario is then the summation of the step costs incurred during the entire simulation (assuming end time $t$):

$$cost = \sum_{j=1}^{t} cost(j) \tag{3}$$

By comparing the step costs and the total cost, we can evaluate the performance of each partitioning mechanism for a given scenario. Experiments were performed to estimate real values for the various associated costs: communication, computation 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, 2.00GB of RAM*. The computers are connected via *100MB Ethernet*. The distributed system will be *HLA/RTI* (Kuhl et al. 1999) compliant and implemented using the DMSO *RTI1.3NG-V6*. The communication cost and migration cost are estimated using a test case with two computers each running a federate. With HLA/RTI their is an associated synchronization overhead at each time step due to *time advance requests*. To approximate this we configure a test case with two servers and no messages and calculate the time needed to execute one hundred time advance requests. To estimate the communication cost, a simulation with two machines is setup and one hundred messages are passed between them. This synchronization cost is then 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 under lying network infrastructure (we assume that with buffering sending one hundred messages from RTI 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. This means the ownership transfer between federates is 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 2.

Table 2: Calculated costs

| Metric | Cost(ms) |
|---|---|
| Computation $C_\alpha$ | 8.074 |
| Communication $C_\beta$ | 6.02 |
| Migration $C_\gamma$ | 51.01 |

We assume a simple agent architecture with little in the way of deliberation. However, the analyzer can easily be adapted to evaluate situations with much heavier weight agents. The importance of the costs is not their absolute values but the relative weighting between each. Table 2 shows that for this case, migration is fifty times more expensive than communication and a little over six times more expensive that computation. It is the relative costs of each metric which determines the effectiveness of each partitioning algorithm.

To validate this and previous work (Wang et al. 2009) we have compared the results generated from the analyzer with real experimental results. To do this we tested the two aforementioned partitioning algorithms: FALB and static partitioning in various configurations and compared the results of a real implementation with that predicted by the analyzer. The test case considered was of 200 agents moving within a square environment of 100x100. Each of the agents was initialized with a random location within the environment and moved from left to right with a fixed velocity of $(0.5, 0)$. Once the $x$ location of an agent reaches 100 it is moved back to the center, $x = 50$. The simulation is run over 200 time steps with 2 and 4 nodes and the total execution time is then recorded. This configuration 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 within the system due to synchronization. The real system was therefore first tested without any agents or communication to calculate the synchronization overhead. This overhead was then subtracted from the result obtained from the real system for direct comparison with the analyzer. This overhead is fixed for all cases so should not invalidate comparisons made between two algorithms using the analyzer.

Figure 2 shows the comparison between what the analyzer predicts and the real system output. It clearly shows that the analyzer prediction is very close to the output of the real system. Figure 5 shows how the analyzer prediction compares in the case of the FALB with four nodes. Again the results are very similar. However, in all cases the analyzer does slightly under-predict, which is to be expected as the analyzer does not consider the overhead associated with execution of the partitioning algorithms. The key issue, which the results corroborate, is that the analyzers predictions follow the overall trend of the real system for a variety of different parameters.

### 6.3 K-Means Test Results

In the following section we use the analyzer tool to investigate the k-means clustered partitioning algorithm described in Section 5. The experimental setup is similar to the validation case except with more agents. The experiment consists of 500 agents executing for 200 time steps in an environment of 100x100. In all cases agents are created at randomly

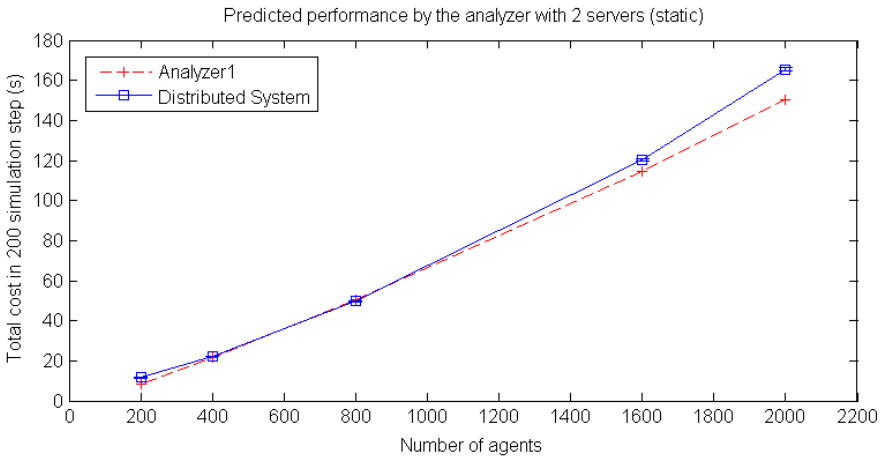


Figure 2: Analyzer Prediction and Real System Output for Static Algorithm and 2 Nodes

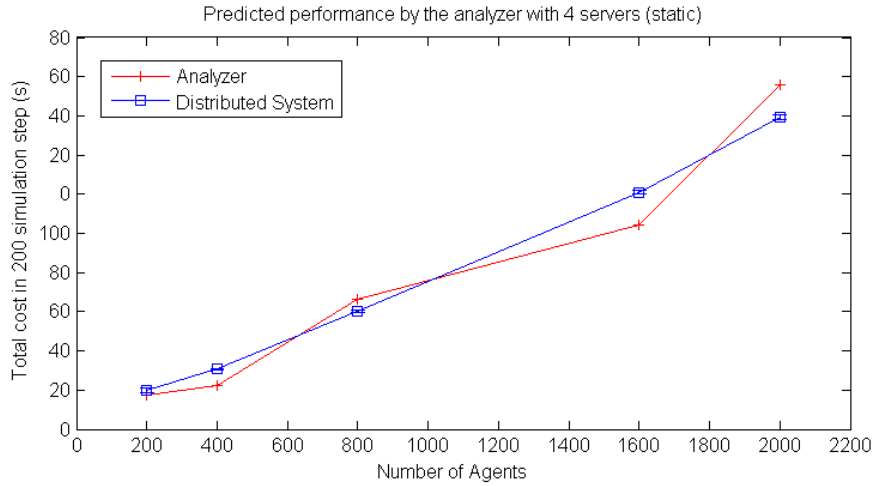


Figure 3: Analyzer Prediction and Real System Output for Static Algorithm and 4 Nodes

initialized positions within the environment. However, the crowd movement is varied using grouped movement and random movement. Under grouped movement a number of leaders are defined which in turn defines the number of groups, the remaining agents are then assigned to groups evenly. The group members follow the leaders while maintaining a minimum distance. The experiments also consider a standard situation and an emergency situation. Under normal circumstances the groups or individuals move randomly. In grouped movement it is the leaders who move randomly and the group members follow. In the emergency situation the agents move randomly until time step 5 at which point a bomb is detonated and the crowd subsequently attempts to escape via a single exit. This gives four test cases for comparison: Random_normal, Random_emergency, Group_normal and Group_emergency.

The results presented are execution times calculated using the analyzer. Figure 6 shows the average total simulation time (over 5 different runs of 100 time steps) predicted by the analyzer for each partitioning mechanism. In Random_normal test case, these three partitioning strategies give almost identical performance. Since the loads on the servers remain in balance for most of the time after initialization, the FALB strategy would offer no performance benefit over the static partitioning strategy. Since no clustering occurs in this scenario, the k-means strategy will not achieve any significant benefit either. Under the Normal_emergency test case, k-means provides the best performance among the three strategies. Once the bomb is detonated, instead of wandering around, individuals will move towards the same goal to evacuate via the exit. This will cause the agents to group together, something which can be exploited by the k-means partitioning algorithm. Since individuals

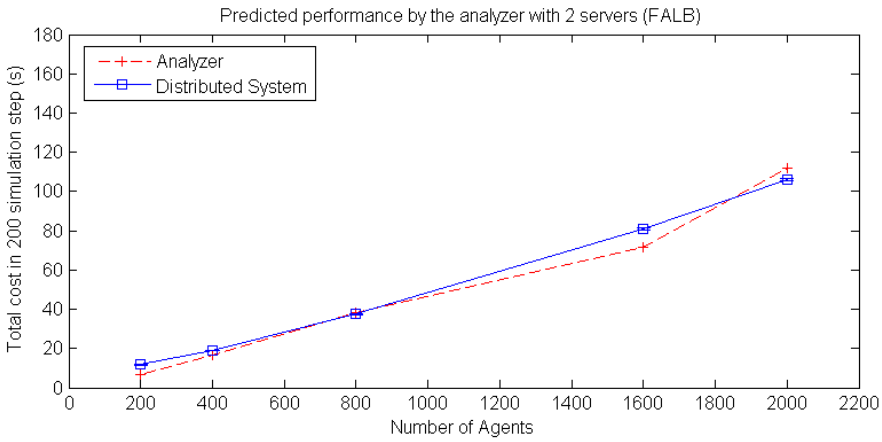


Figure 4: Analyzer Prediction and Real System Output for FALB Algorithm and 2 Nodes

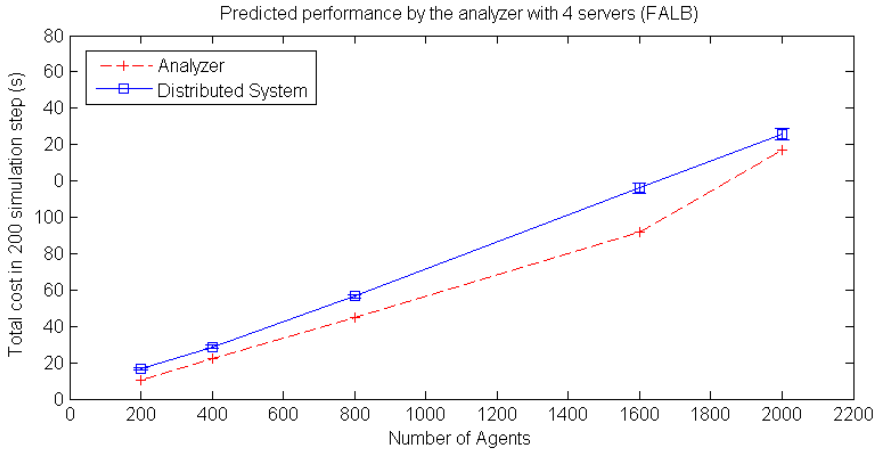


Figure 5: Analyzer Prediction and Real System Output for FALB Algorithm and 4 Nodes

share the same goal, the movement of each formed cluster is paced which results in less migrations between partitions. For FALB, the boundaries of each partition will be adjusted in order to balance the load in each partition. Migration can therefore occur in two ways: through movement of the partition boundaries and by agents transitioning between two partitions. In this test case, the static strategy faces extreme load imbalance which is why it has poor performance.

In both Group_normal and Group_emergency test cases, k-means provides the best performance among the three algorithms. Since the composition of each cluster is more or less constant - with the exception of when individuals join or leave the group - the migration cost is low. At the same time, maintaining the clustering also means the local states are kept within in each partition, which reduces the communication as well. Note that, although the number of groups in the experiments is the same as the number of k-means clusters the algorithm still performs well when the grouping in the experiment changes (i.e., during the emergency). The FALB algorithm is not suitable for grouped scenarios as it tends to emphasize balancing load rather than minimizing communication cost. Figure 7 shows a case that the FALB strategy divides a group into two different servers (the lines depict the division of the clusters). In this scenario the dark group (top right) is divided across two partitions. If the partition boundary is changed slightly or whenever the group moves there will be frequent migrations. While one may consider a grouped scenario to be engineered toward a clustering approach, we argue that grouping is relevant for crowd simulation. The results also show that for random configurations the k-means algorithm performs equivalently to a standard static approach. Perhaps more interestingly, the emergency scenarios are important for crowd simulation and DVEs in general. In previous work (Greenhalgh 1997) the notion of *Hot points* was introduced to describe the notion of

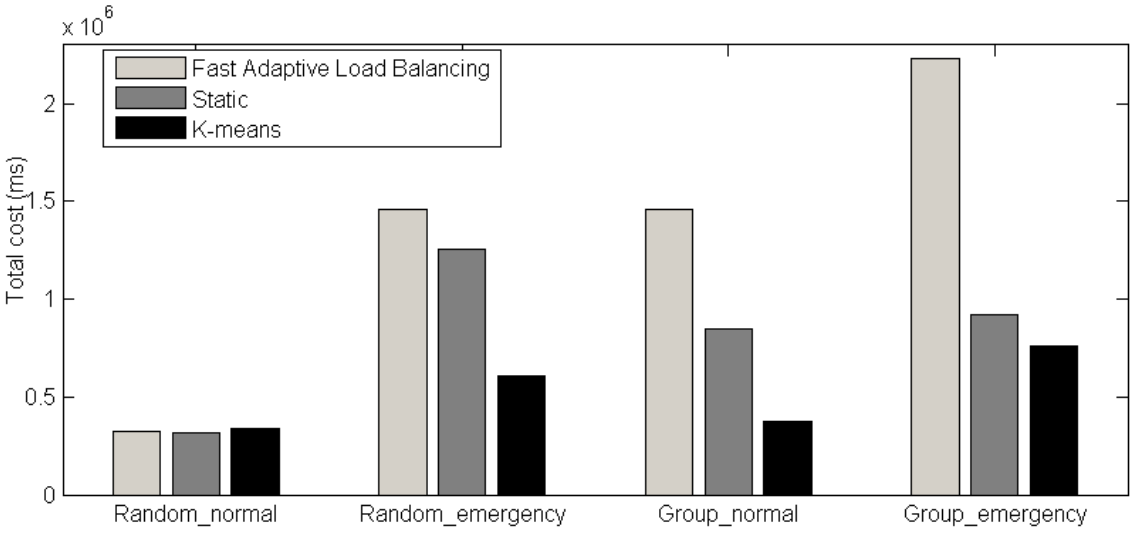


Figure 6: Performance Comparison of Different Partitioning Strategies

areas of the environment where avatars or agents would congregate. If partitioning is done by environment (e.g., static or FALB) the areas around these hot points will typically be handled by a single server. With many agents clustered around these areas and the servers that maintain them will have high load imbalance. It is these scenarios in which we expect the clustering approaches to work best.

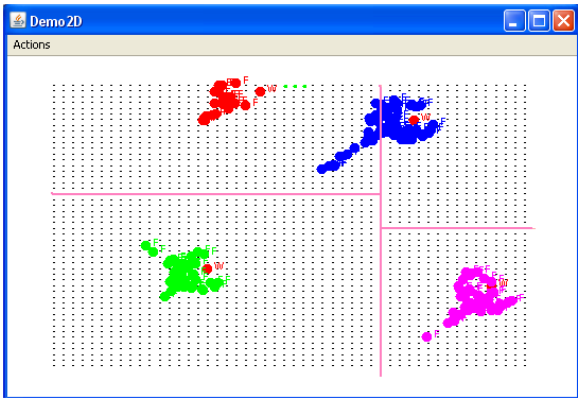


Figure 7: Partitions under Group_normal scenario

Also because the group is divided across the two servers, there will be a large amount of communication among the servers. Both the static and FALB strategies have more migrations than the k-means when individuals cross the boundaries of each partition. The Group_emergency scenario shows similar results for the same reason as above, again with the k-means out-performing both of the other algorithms.

## 7 CONCLUSION AND FUTURE WORK

In this paper we have motivated the application of cluster partitioning algorithms for agent-based crowd simulation. Using an analyzer tool we have shown that for some typical crowd scenarios where groups form, our k-means load-balancing algorithm outperforms other approaches. We have also validated the use of our analyzer and shown it to be an effective tool for efficiently comparing different load balancing and partitioning algorithms. The only current limitation of the analyzer is that it does not include the overhead associated with the execution of each algorithm. Incorporating this directly into the

tool would be difficult but we plan to investigate ways of manually including an approximated overhead for each tested algorithm.

In future work we plan to investigate other cluster based partitioning mechanisms (grid-based clustering) and compare their performance to the k-means algorithm. One issue with cluster based approaches is how the crossing problem (see Figure 1) can be solved. If groups (or clusters) form intermittently, the methods which automatically cluster agents may cause unnecessary migration. Therefore, we hope to look at applying pattern recognition techniques to try and analyze and predict crowd movement to aid partitioning.

## REFERENCES


Ali, S., and M. Shah. 2007. A lagrangian particle dynamics approach for crowd flow segmentation and stability analysis. In *IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR '07.*, 1–6.

Dechter, R., and J. Pearl. 1985. Generalized best-first search strategies and the optimality of A*. *J. ACM* 32 (3): 505–536.

Deng, Y., R. F. Peierls, and C. Rivera. 2000. An adaptive load balancing method for parallel molecular dynamics simulations. *J. Comput. Phys.* 161 (1): 250–263.

Engenuity Technologies Inc 2006. Ai.Implant. http://www.ai-implant.com.

Floyd, R. W. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5 (6): 345.

Greenhalgh, C. 1997. Analysing movement and world transitions in virtual reality tele-conferencing. In *ECSCW'97: Proceedings of the fifth conference on European Conference on Computer-Supported Cooperative Work*, 313–328. Norwell, MA, USA: Kluwer Academic Publishers.

Gürsoy, A. 2003. Data decomposition for parallel k-means clustering. In *PPAM*, ed. R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Wasniewski, Volume 3019 of *Lecture Notes in Computer Science*, 241–248: Springer.

Kuhl, F., R. Weatherly, and J. Dahmann. 1999. *Creating computer simulation systems: An introduction to the high level architecture*. Prentice Hall PTR.

Lees, M., B. Logan, and J. King. 2007, June. Hla simulation of agent-based bacterial models. In *Proceedings of the 2007 European Simulation Interoperability Workshop*. Genoa: Simulation Interoperability Standards Organisation. 07E-SIW-032.

Lees, M., B. Logan, and G. Theodoropoulos. 2007. Distributed simulation of agent-based systems with HLA. *ACM Transactions on Modeling and Computer Simulation* 17 (3): 11.

Logan, B., and G. Theodoropoulos. 2001, February. The distributed simulation of multi-agent systems. *Proceedings of the IEEE* 89 (2): 174–186.

Lozano, M., P. Morillo, D. Lewis, D. Reiners, and C. Cruz-Neira. 2007. A distributed framework for scalable large-scale crowd simulation. In *HCI (14)*, 111–121.

Luo, L., S. Zhou, W. Cai, M. Y. H. Low, F. Tian, Y. Wang, X. Xiao, and D. Chen. 2008. Agent-based human behavior modeling for crowd simulation. *Comput. Animat. Virtual Worlds* 19 (3-4): 271–281.

Marzouk, Y. M., and A. F. Ghoniem. 2005. K-means clustering for optimal partitioning and dynamic load balancing of parallel hierarchical n-body simulations. *J. Comput. Phys.* 207 (2): 493–528.

Massive Software 2007. Massive. http://www.massivesoftware.com.

Morillo, P., n. J. M. Ordu and M. Fernández. 2004. A comparison study of evolutive algorithms for solving the partitioning problem in distributed virtual environment systems. *Parallel Computing* 30 (5-6): 585–610.

Musse, S. R., and D. Thalmann. 2001. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics* 7 (2): 152–164.

Nguyen, Q.-A. H., F. D. McKenzie, and M. D. Petty. 2005. Crowd behavior architecture model cognitive design. *Proceedings of the 2005 Conference on Behavior Representation in Modeling and Simulation (BRIMS)*:55–64.

Oguara, T., D. Chen, G. Theodoropoulos, B. Logan, and M. Lees. 2005, October. 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)*, ed. A. Boukerche, S. J. Turner, D. Roberts, and G. Theodoropoulos, 179–186. Montreal, Quebec, Canada: IEEE Press.

Pan, X., C. S. Han, and K. H. Law. 2005. A multi-agent based simulation framework for the study of human and social behavior in egress analysis. *The International Conference on Computing in Civil Engineering*:12–15.

Paris, S., S. Donikian, and N. Bonvalet. 2006. Environmental abstraction and path planning techniques for realistic crowd simulation. *Computer Animation and Virtual Worlds* 17 (3-4): 325–335.

Pettre, J., P. de Heras Ciechomski, J. Maim, B. Yersin, J.-P. Laumond, and D. Thalmann. 2006. Real-time navigating crowds: scalable simulation and rendering. *Computer Animation and Virtual Worlds* 17:445–455.

Santos, G., and B. E. Aguirre. 2004, June. A critical review of emergency evacuation simulation models. *NIST Workshop on Building Occupant Movement during Fire Emergencies*:25–50.

Sung, M., M. Gleicher, and S. Chenney. 2004. Scalable behaviors for crowd simulation. *Computer Graphics Forum* 23 (3): 519–528.

van den Berg, J., M. Lin, and D. Manocha. 2008, May. Reciprocal velocity obstacles for real-time multi-agent navigation. *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008.*:1928–1935.

Wang, Y., 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-base simulation systems. *the 40th IEEE Annual Simulation Symposium (ANSS)*.

Zhang, D., C. Jiang, and S. Li. 2009. A fast adaptive load balancing method for parallel particle-based simulations. *Simulation Modelling Practice and Theory* In Press, Accepted Manuscript.

## AUTHOR BIOGRAPHIES

**YONGWEI WANG** is currently a Project Officer in the School of Computer Engineering at Nanyang Technological University (NTU), Singapore. He is a part-time Master candidate and received his B.Eng. in Computer Engineering from NTU. His Master project is on the distributed simulation system. His research interests include: distributed computation, agent-based simulation, and high level communication architecture. His email is <david.wang@ntu.edu.sg>.

**MICHAEL LEES** is a Post-doctoral Research Fellow with the School of Computer Engineering at Nanyang Technological University (NTU), Singapore. Previously he has held research positions both at the University of Nottingham (UK) and the University of Birmingham (UK). He obtained his B.S.c. from the University of Edinburgh in 2001 and his Ph.D. from the University of Nottingham in 2005. His research interests include agent-based systems, modeling and simulation, distributed simulation and high performance computing. His email is <mhlees@ntu.edu.sg>.

**WENTONG CAI** is an Associate Professor with the School of Computer Engineering at Nanyang Technological University (NTU), Singapore, and head of the Computer Science Division. He received his Ph.D., in Computer Science, from University of Exeter (U.K.). He was a Postdoctoral Research Fellow at Queens University (Canada) before joining NTU in February 1993. Dr. Cais research interests include Parallel and Distributed Simulation, Grid and Cluster Computing, and Parallel and Distributed Programming Environments. His main areas of expertise are the design and analysis of scalable architecture, framework, and protocols to support parallel and distributed simulation, and the development of models and software tools for programming parallel/distributed systems.He has authored or co-authored over 180 journal and conference papers in the above areas. Dr. Cai is a member of the IEEE. He is currently an associate editor of ACM Transactions on Modeling and Computer Simulation (TOMACS), editorial board member of Multiagents and Grid Systems -An International Journal, and editorial board member of International Journal of Computers and Applications. His email is <ASWTCAI@ntu.edu.sg>.

**SUIPING ZHOU** is currently an Assistant Professor in the School of Computer Engineering at Nanyang Technological University (Singapore). Previously, he worked as an engineer in Beijing Simulation Center, China Aerospace Corporation, and then joined Weizmann Institute of Science (Israel) as a Post-doctoral fellow. He received his B.Eng., M.Eng. and Ph.D in Electrical Engineering from Beijing University of Aeronautics and Astronautics (P.R. China). He is a member of IEEE and his current research interests include: large-scale distributed interactive applications (e.g., MMOGs), parallel/distributed systems, and human behaviour representation in modeling and simulation. He has published over 60 peer reviewed articles in these areas. He is currently an associate editor of the International Journal of Computer Games Technology. He has served as technical program committee member of many international conferences and workshops in computer games and virtual environments. His email is <ASSPZhou@ntu.edu.sg>.

**MALCOLM YOKE HEAN LOW** is an Assistant Professor in the School of Computer Engineering at the Nanyang Technological University (NTU), Singapore. Prior to this, he was with the Singapore Institute of Manufacturing Technology, Singapore (SIMTech). He received his Bachelor and Master of Applied Science in Computer Engineering from NTU in 1997 and 1999 respectively. He was awarded a Gintic (now SIMTech) Postgraduate Scholarship in 1999. In 2002, he received his D.Phil degree in Computer Science from Oxford University. His current research interest is in the application of parallel and distributed computing for the modeling, simulation, analysis and optimization of complex systems. His email is <YHLow@ntu.edu.sg>.