Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

GRID-BASED PARTITIONING FOR LARGE-SCALE DISTRIBUTED AGENT-BASED CROWD SIMULATION

Yongwei Wang
Michael Lees
Wentong Cai

School of Computer Engineering,
Nanyang Technological University,
Singapore

ABSTRACT

Agent-based crowd simulation, which aims to simulate large crowds of autonomous agents with realistic behavior, is a challenging but important problem. One key issue is scalability. Parallelism and distribution is an obvious approach to achieve scalability for agent-based crowd simulation. Parallel and distributed agent-based crowd simulation, however, introduces its own challenges, in particular, effectively distributing workload amongst multiple nodes with minimal overhead. In order to ensure effective distribution with low overhead, a proper partitioning mechanism is required. Generally, human crowds consist of groups or exhibit particular patterns of flow, which are then reflected in simulations. Exploiting this grouping with an appropriate partitioning mechanism should enable efficient distribution of crowd simulation. In this paper we introduce a grid-based clustering algorithm which we compare to previous clustering approaches that used the K-means algorithm.

1 INTRODUCTION

The simulation of large crowds has become an essential tool for many virtual environment applications in education, training, and entertainment. However, simulating the realistic behavior of large crowds is still a challenge for several computer research communities. 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 individual level. It provides a higher degree of heterogeneity in terms of the behavioral modeling.

Agent-based crowd simulation uses a microscopic approach and has been adopted by many research groups (Crowd Dynamics 2012; Guye-Vuillieme and Thalmann 2001; Petty, McKenzie, Gaskins, and Weisel 2004). For agent-based crowd simulation, each individual in a crowd is modeled as an intelligent agent, which may have various capabilities to behave in the simulated world, ranging from reacting to triggered events to adapting to complex dynamic environments. Autonomous individuals, or more precisely agents, are regulated by sets of decision rules and each agent can make independent decisions. The agent will typically perceive its local environment and obtain relevant information within its area of interest (AOI). Heterogeneous behaviors of individuals may be achieved by tuning the personal attributes and behavioral characteristics of agents. Each agent operates within a cycle, performing sense, think, and act procedures as a means of achieving some specified tasks. One can argue that the agent-based approach is the most natural way to model behaviors with strong individual differentiation. For this reason the agent-based methodology is perhaps the most popular modeling mechanism currently used in the crowd simulation

978-1-4673-4781-5/12/$31.00 ©2012 IEEE

community. However, for applications that attempt to simulate large-scale crowds, the computational resources necessary for an agent-based model can be significant.

Parallel and distributed simulation is one natural approach to improve the scalability of agent-based models 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 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 inherent 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. Time-stepped parallel or distributed execution typically involves periods of concurrent 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, to achieve maximal benefit and speed up from distribution, an effective load balancing mechanism is essential.

However, distribution itself introduces further issues. 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 (Bezerra, Cecin, and Geyer 2008; Minson and Theodoropoulos 2005; Morgan, Lu, and Storey 2005) 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 this is typically associated with the entities' locality. The key assumption being that those entities nearby to each other are considered to be interested in one another and therefore need to exchange state information. Therefore, by partitioning entities which interact frequently onto the same computing node, communication overhead will be reduced.

The problem begins with a distributed collection of processors which when put together, form a single virtual environment (or computational space). The challenge lies on 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 important factor that determines the effectiveness of a parallel and distributed simulation. Migration involves the transfer of entities from one partition to another, often migration is employed to bring entities that communicate frequently onto the same processor. 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 consistency, especially when a complex agent model with large internal state is adopted. There is obviously a trade-off that must be made between migration and communication. 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. If it is possible to know how the agents are grouped now and how these groups will change with time, this information can be exploited to develop highly effective partitioning algorithms.

In this paper, we propose a new clustering algorithm which uses the grid-based clustering technique. In contrast to previous work, the algorithm attempts to consider how agents positions change with time and use this information to intelligently guide the partitioning process. Several experiments are designed to investigate the strengths and weaknesses of this new algorithm in relation to previous approaches (Wang, Lees, Cai, Zhou, and Low 2009). The remainder of this paper is organized as follows: section two introduces

related work in partitioning for agent-based simulation and work which has employed clustering methods in distributed simulation. Section three continues by describing the principle of cluster-based partitioning and the details of grid-based algorithm. Section four presents the experimental evaluation and highlights the suitability of both approaches. The final section presents the conclusions and discussion.

2 RELATED WORK

Typically, there are two different approaches for partitioning an agent-based crowd simulation. One approach is region-based where the agent's 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.

Many distributed multi-agent systems adopt a static environment decomposition (for example (Lees, Logan, and King 2007; Lees, Logan, and Theodoropoulos 2007)). This involves a simple form of region-based partitioning. In some agent systems this may be sensible approach, 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.

In (G. Vigueras and Grimaldo 2010), the authors investigated region-based partitioning methods for crowd simulation and compare their performance. They define a fitness function $H(P)$ to evaluate the performance of a partitioning method.

$$H(P) = \omega_1 * \alpha(P) + \omega_2 * \beta(P), \quad \omega_1 + \omega_2 = 1 \tag{1}$$

$\alpha(P)$ 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 region-based partitioning methods by using the 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. They fail to consider that agents may migrate from one partition to another between two static states.

In this paper, we aim to apply the clustering techniques to partition the crowd simulation. We aim to balance the computational load, minimize the communications and migrations at the same time. 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 previous work (Wang, Lees, Cai, Zhou, and Low 2009) we presented a partitioning algorithm for agent-based crowd simulation which used K-means clustering. The key difference between the new approach and the previous approach, apart from the change in method, is the ability of the algorithm to utilize information about the future movement of the crowd (using goals). This way we hope that the grid-based approach will be able to more complex patterns of interaction without suffering from large amounts of agent migration. The second key benefit of the grid-based approach is that it lends itself far more easily to a parallel implementation of the algorithm.

3 GRID-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 in agent-based crowd simulation 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 can be effectively 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. In general, unlike N-body problems (Aarseth 2003) (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.

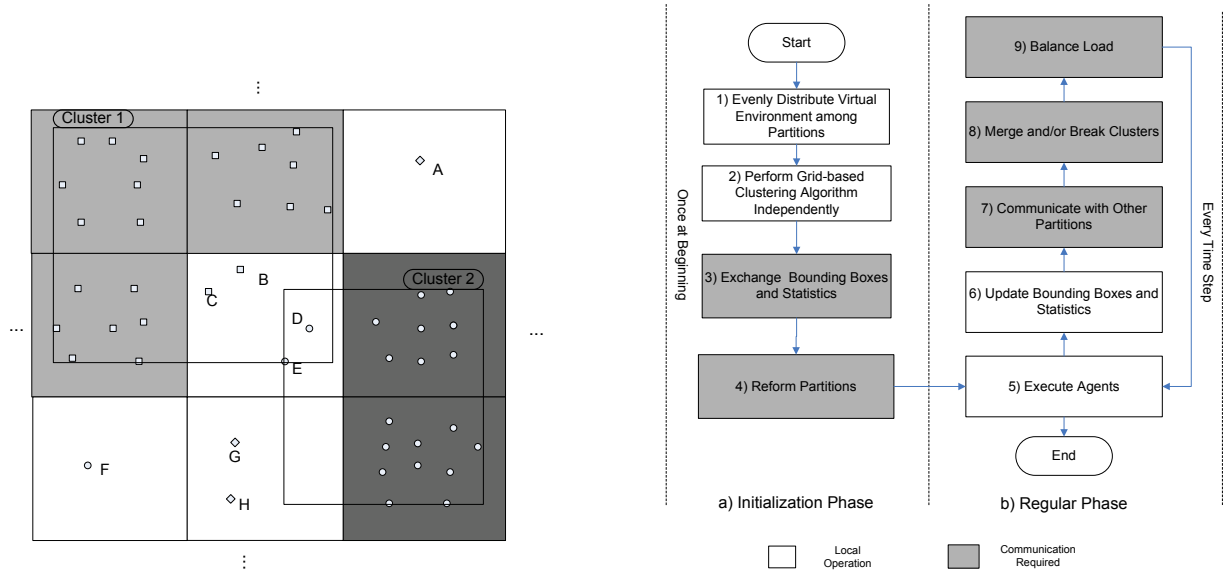
3.1 Concepts

In this paper we present a new clustering based approach for partitioning, which uses a grid-based methodology. When compared with other clustering methods, the grid-based clustering is known for its fast processing time (Liao, Liu, and Choudhary 2004). The algorithm also offers a great deal of control as the execution time is proportional to the selected grid size rather than the number of data objects. grid-based 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, these statistics are then used to 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 to allow for more accurate and balanced partitions.

To ease explanation we point out the distinction between the grid-based clustering algorithm and the grid-based partitioning algorithm. The former is the method described above and obtained from literature, the later refers to the overall computational algorithm which we propose in this paper. Before we further investigate the grid-based partitioning method, it is helpful to clarify terminology. Figure 1(a) 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 in 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 most common goal. In our study, g_{var} is considered as high if it is greater than 10% (which we obtained through experimentation, in general this is a parameter).



(a) Simple Grid-based Partitioning Demonstration

(b) Steps of Grid-based Partitioning Algorithm

Figure 1: Grid-based partitioning.

Cells with high goal position variance (for example the center cell in Figure 1(a)) 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 1(a), 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*. 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 in identifying 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, which is identified dynamically by the algorithm, may not coincide with the number of servers available. We need to further place the identified clusters and individuals into partitions. In the remainder of this section, we will discuss our distributed grid-based partitioning algorithm with considerations of load balancing, interest management and migrations.

3.2 Algorithm

The 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 1(b) 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 initialization phase, the virtual environment is evenly

distributed among partitions so that each partition contains equal area of the virtual environment. Each partition (or computational node) 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 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. 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 also $O(N_{partition})$.

The final step is the process of clustering the similar cells recursively by traversing grid cells. Beginning with the top left cell, the cell merging process attempts to cluster neighbor cells based on their common goals statistic. Once a neighbor cell with the same common goal position is identified, the algorithm will include the neighbor cell in the 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. 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.

3) Exchange Bounding Boxes and Statistics: The configurations (minimum and maximum x and y coordinates) of all the bounding boxes in a partition are broadcast among the other partitions. The statistics of each cluster are broadcast as well in order to reform partitions.

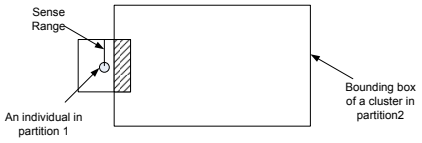
4) Reform Partitions: If two clusters are located in different partitions but have overlapping bounding boxes and the same common goal position, a merge of these two clusters will occur. The merged cluster will be assigned to the partition (amongst all the overlapping partitions) with the least agents. 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, the algorithm calculates 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, the furthest cluster(s) from the partition center are re-assigned 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 clusters 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 of the COSMOS project (Luo, Zhou, Cai, Lees, Low, and Sornum 2011) is used. Agents sense the virtual world and nearby agents, they then filter the perceived information and make decisions based upon that information. Their decisions cause the agents to move around in the virtual environment according to the decided goals. As a result, we update agents' positions, inner states, perceptions, and decisions

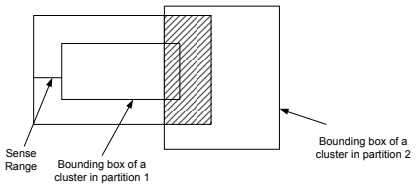
periodically during simulation. The partitions will communicate with each other to retrieve necessary states of agents and virtual environment for the execution of perceptions (see step 7). An interest management algorithm is required to perceive the virtual environment and nearby agents with low communications.

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

7) Communicate with Other Partitions: The communication among partitions involves exchanging states for the interest management and load balancing. In our research we adopt the bounding box concept to define what information needs to be communicated between partitions. To enable the use of the bounding box to define interest, it is necessary that the bounding box coordinates for the clusters, as well as the statistics of each cluster, are broadcast among all partitions at each time step.



(a) Interest Management between Individual and Clusters



(b) Interest Management between Clusters

Figure 2: Interest Management Methods.

In our design, an individual has a rectangular 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 those necessary for individuals or agents in a cluster to sense and make decisions. Figure 2(a) and Figure 2(b) show two cases in which a partition needs to request for agents' states from other partitions. In Figure 2(a), the individual from partition 1 has an extended bounding box which overlaps with 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 updates of the agents which lie in the intersection area to partition 1. Likewise, in Figure 2(b), 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 then 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 (see step 4). 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. 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. If 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 distribution of agents among partitions 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. The partition will try to shed those clusters and individuals which are furthest from the cluster center. Conversely, 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 the *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 1(b), until the simulation stops.

4 EXPERIMENTS

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. Again, an agent is assumed to interact with other agents when their AOIs overlap. To minimize the interactions and communication, 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 be assigned to different partitions. This in turn may result in a large amount of communication 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.

To evaluate the effectiveness of these algorithms we use a load balancing analyzer, which we have previously introduced and validated (Wang, Lees, Cai, Zhou, and Low 2009; Wang, Cai, Low, Zhou, Tian, Luo, Ong, and Hamilton 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. The details of the parameters and setup of the analyzer are omitted here due to space limitations, however details can be found in previous work (Wang, Lees, Cai, Zhou, and Low 2009).

4.1 Experimental 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 case studies in our research. That is, the 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 (Reynolds 1987). m group leaders are selected at random and then $(n \times g) - m$ agents are assigned in turn to follow 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. Note that the primary focus of these experiments is to compare the relative performance of the two algorithms. Existing, more detailed analysis of the K-means approach can be found in our previous work (Wang, Lees, Cai, Zhou, and Low 2009). As the scenarios are engineered to evaluate particular performance characteristics of the algorithms, the total number of timesteps is chosen so as to sufficiently capture the relevant dynamics. The same applies for the number of agents used. In general, given the same number of compute nodes, we would expect the performance of the algorithms to be worse with fewer total agents. Of course crowd simulations vary in size dramatically, ranging from a few hundred individuals up to tens of thousands. We opt for reasonable numbers of agents, which we expect to illustrate the performance of the algorithms in the more demanding but realistic configurations.

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 agents' movement so that the algorithms will be able to determine appropriate clusters.

4.2 Case Study 1 - Effect of Grouping

The first case study considers a simple rectangular environment of 100mx100m. 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.

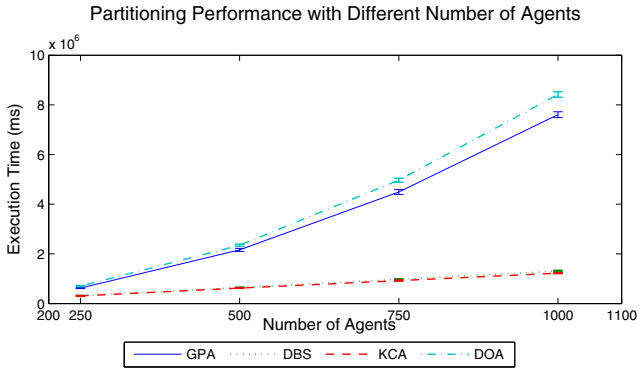


Figure 3: Partitioning Performance for Different Number of Agents.

Figure 3 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 remain unchanged during simulation since no cluster pattern is shown in this case study. As a result, agents near to each other may not be on the same partition during the 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 move randomly and individually, a static partitioning of the environment is a natural option.

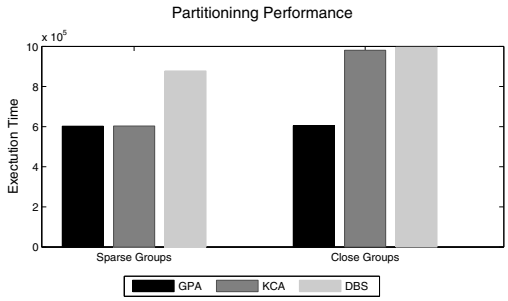
Table 1: The Execution Time for Different Partitioning Algorithms. Number of Agents n = 1000 & Group % g = 1 & Group Num m =8.

	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

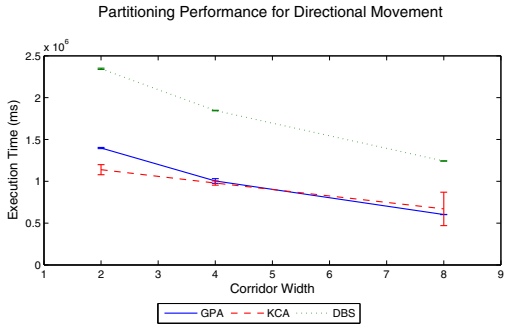
The DOA algorithm offers the worst performance when group patterns are shown in the simulation. For instance, Table 1 compares the execution time for these four partitioning algorithms in a simulation scenario with 1000 agents. All the agents are formed in 8 groups. The table shows that the execution 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.

4.3 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 the same open square virtual environment but with predefined routes.



(a) Partitioning Performance for Circular Movement



(b) Partitioning Performance for Different Corridor Widths (including confidence intervals)

Figure 4: Experiment Two.

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. Two experiments are designed, namely sparse groups and close groups. 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. The groups in the same set are located near to each other but further away from the groups in the other set. Figure 4(a) shows the partitioning performance for different partitioning methods for the two experiments. For the first experiment, the GPA and KCA exhibit more or less identical performance. The execution time for the GPA and KCA is about two thirds that of static partitioning. These two algorithms take into consideration the dynamic movement of the groups and the partitions manage to 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 the same. In the second experiment, the KCA may misclassify agents. It may divide a naturally formed group into two partitions and introduce more communication and migration.

We also developed experiments using a virtual environment with a narrow corridor where agents have directed motion to investigate the problem of group split and merge. With all agents moving along a fixed path, one would expect that the clustering algorithms utilize this information to partition the agents. The size of narrow corridor is important in evaluating the performance of different algorithms. In the experiment we vary the corridor width (4m, 4m and 8m). Again, there are 1000 agents in the scenario. Agents are divided into four groups and group leaders will lead their group. The four groups are divided into two sides, with two groups initialized at the upper side and the 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.

Figure 4(b) 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. When the corridor width equals 8m, groups are able to navigate through the corridor without overlapping. The results show that the GPA offers better performance than the KCA for wider corridors. Recall the KCA clusters agents based on their locations to the centroids, where the 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 closer to the centroid of the other group. As a result, agents in a natural group could be divided into two distinct clusters thereby introducing communication between clusters. In this situation, the GPA should provide a better performance than the KCA because the GPA clusters agents based on their locations as well as future goals. If two groups of agents have distinct enough goals, they will remain in their respective clusters even when two groups overlap with each other. However, when two groups overlap, the results indicate that it may be better to repartition agents according to locations to minimize the communication cost. In such a 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 become more distinct from each other. The wider corridor reduces the amount of communication for all three partitioning algorithms. The GPA and KCA provide significant improvement over the DBS at they are able to adapt to the dynamic movement of the groups and maintain a balanced load among computation nodes. In contrast, the DBS cannot ensure a balanced partitioning.

5 CONCLUSIONS

In this paper we examined the problem of load balancing for distributed agent-based crowd simulation and discussed the importance of interest management for effective partitioning algorithms. We presented and compared two partitioning algorithms, K-means and grid-based. We examined the effectiveness and performance of the K-means and grid-based partitioning algorithms for a number of typical crowd simulation scenarios. 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 within the crowd. As the K-means partitioning algorithm clusters agents solely based on their locations, it suffers from a misclassification problem, which becomes worse when there are individuals (outlying agents) in the simulation and when two naturally formed groups move closer 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).

REFERENCES

Aarseth, S. J. 2003. *Gravitational N-body Simulations: Tools and Algorithms*. Cambridge University Press.

Bezerra, C. E., F. R. Cecin, and C. F. R. Geyer. 2008, Oct.. "A3: A Novel Interest Management Algorithm for Distributed Simulations of MMOGs". In *Proceedings of Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium*, edited by D. Roberts, A. El-Saddik, and A. Ferscha, 35–42.

Crowd Dynamics 2012. "Crowd Dynamics". Accessed Sept. 30 2012. http://www.crowddynamics.com.

G. Vigueras, M. Lozanoand, J. M. O., and F. Grimaldo. 2010. "A Comparative Study of Partitioning Pethods for Crowd Simulations". *Applied Software Computing* 10 (1): 225–235.

Guye-Vuillieme, A., and D. Thalmann. 2001. "A High Level Architecture for Believable Social Agents". *Virtual Reality Journal* 5:95–106.

Lees, M., B. Logan, and J. King. 2007, Jun. "HLA Simulation of Agent-Based Bacterial Models". In *Proceedings of the 2007 European Simulation Interoperability Workshop*. Genoa: Simulation Interoperability Standards Organisation: 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.

Liao, W. K., Y. Liu, and A. Choudhary. 2004, April. "A Grid-based Clustering Algorithm Using Adaptive Mesh Refinement". 61–69. Lake Buena Vista, Florida, USA: *7th Workshop on Mining Scientific and Engineering Datasets in conjunction with SIAM International Conference on Data Mining (SDM)*.

Luo, L., S. Zhou, W. Cai, M. Lees, M. Low, and K. Sornum. 2011. "HumDPM: A Decision Process Model for Modeling Human-Like Behaviors in Time-Critical and Uncertain Situations". In *Transactions on Computational Science XII*, Volume 6670 of *Lecture Notes in Computer Science*, 206–230. Springer Berlin / Heidelberg.

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.

Minson, R., and G. Theodoropoulos. 2005, June. "An Adaptive Interest Management Scheme for Distributed Virtual Environments". In *Proceedings of Principles of Advanced and Distributed Simulation, 2005. PADS 2005. Workshop*, 273–281.

Morgan, G., F. Lu, and K. Storey. 2005. "Interest Management Middleware for Metworked Games". In *I3D '05: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, edited by A. Lastra, M. Olano, D. P. Luebke, and H. Pfister, 57–64. New York, NY, USA: ACM.

Petty, M. D., F. D. McKenzie, R. C. Gaskins, and E. W. Weisel. 2004. "Developing a Crowd Federate for Military Simulation". 483–493.

Reynolds, C. W. 1987. "Flocks, herds and schools: A distributed behavioral model". In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, 25–34. New York, NY, USA: ACM.

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-based simulation systems". In *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, 33:1–33:8. San Diego, CA, USA: Society for Computer Simulation International.

Wang, Y., M. Lees, W. Cai, S. Zhou, and M. Y. H. Low. 2009, December. "Cluster Based Partitioning for Agent-Based Crowd Simulations". In Proceedings of the 2009 Winter Simulation Conference.

AUTHOR BIOGRAPHIES

YONGWEI WANG was a Project Officer in the School of Computer Engineering at Nanyang Technological University (NTU), Singapore. He completed his Masters degree and received his B.Eng. in Computer Engineering from NTU. His Master project was on the distributed simulation of human crowds.

MICHAEL LEES is an Assistant Professor in the School of Computer Engineering, Nanyang Technological University (NTU). His research interests are primarily in modelling and simulation of large scale complex systems, he is particularly interested in understanding the effect that human behavior has on such systems and the important role that individual behavioral interactions have on system level dynamics. His email address is mhlees@ntu.edu.sg and his web page is http://www.mhlees.com

WENTONG CAI is a Professor in the School of Computer Engineering at Nanyang Technological University, Singapore. He is also the Director of the Parallel and Distributed Computing Centre. His expertise is mainly in the areas of Modeling and Simulation (particularly, modeling and simulation of large-scale complex systems, and system support for distributed simulation and virtual environments) and Parallel and Distributed Computing (particularly, Cloud, Grid and Cluster computing). His email address is aswwt-cai@ntu.edu.sg and his web page is http://www.ntu.edu.sg/home/aswtcai/.