Heat Diffusion Based Dynamic Load Balancing for Distributed Virtual Environments

Yunhua Deng Rynson W.H. Lau
Department of Computer Science, City University of Hong Kong, Hong Kong

Abstract

Distributed virtual environments (DVEs) are becoming very popular in recent years, due to their application in online gaming and social networking.One of the main research problems in DVEs is on how to balance the workload when a lot of concurrent users are accessing it. There are a number of load balancing methods proposed to address this problem. However, they either spend too much time on optimizing the partitioning process and become too slow or emphasize on efficiency and the repartitioning process becomes too ineffective. In this paper, we propose a new dynamic load balancing approach for DVEs based on the heat diffusion approach which has been studied in other areas and proved to be very effective and efficient for dynamic load balancing. We have two main contributions. First, we propose an *efficient cell selection scheme* to identify and select appropriate cells for load migration. Second, we propose two heat diffusion based load balancing algorithms, local and global diffusion. Our results show that the new algorithms are both efficient and effective compared with some existing methods, and the global diffusion method performs the best.

CR Categories: C.4 [Performance of System]: Performance attributes; I.3.2 [Graphics Systems]: Distributed/network graphics

Keywords: Dynamic load balancing, dynamic repartitioning, distributed virtual environments, heat diffusion.

1 Introduction

In a distributed virtual environment (DVE), a user with access to the Internet can explore a place of interest without having to travel there, or participate in some activities with some other users from different geometrical locations. Applications for DVE systems include multiplayer online games, military and industrial remote training, collaborative engineering, and distance education. As the scale of some DVE systems is becoming very large, multi-server support for DVEs is attracting some research attention in recent years. One popular approach is to split the load to multiple servers by partitioning the virtual environment (VE) into multiple regions, with each region being served by a single server. Traditionally, this partitioning process is treated as an offline process. For example, in [Funkhouser 1995], the VE is statically partitioned so that each region is assigned to a fixed server. The advantages of this offline load balancing approach are that it is simple and the speed of the partitioning process does not affect the performance of the DVE. However, as the users of a DVE system move around inside the VE, some regions may have too many users due to some application objectives while others may have too few users. As a result, the servers serving the crowded regions will become overloaded and the users being served may suffer from significant delay, while other servers may be under-utilized.

To address the above limitation, a different approach is to perform the load balancing process dynamically during runtime. However, an important requirement of DVEs is interactive response. This means that the computational cost of the load balancing process should be as light as possible in order not to affect the interactivity of the DVE system.

One strategy to re-balance the load during runtime is to repeatedly use one of the partitioning algorithms for offline load balancing to compute new partitioning from scratch each time when the load becomes imbalanced. We call this strategy *load balancing through repartitioning from scratch*. There are a few methods that are based on graph partitioning belonging to this strategy. Some of them make use of the spatial information of the nodes to partition the whole graph into a number of subgraphs with roughly the same number of nodes, such as recursive coordinate bisection [Berger and Bokhari 1987], recursive inertial bisection [Simon 1991], scattered decomposition [Morison and Otto 1987], and index-based partitioner [Ou et al. 1996]. Others may make use of the explicit graph information for partitioning, like recursive spectral bisection [Simon 1991], genetic algorithm [Bui and Moon 1996], and simulated annealing method [Williams 1991]. These methods can produce well balanced partitions even if the load becomes significantly imbalanced due to dramatic state changes. However, repartitioning from scratch can be very time consuming. In addition, each round of partitioning may generate partitions that deviate considerably from the old ones, causing very large data redistribution. As such, this strategy is not suitable for addressing the load balancing problem in DVEs.

An alternative strategy is to migrate the load from the servers with heavier load to the servers with lighter load, effectively shifting the partition boundaries to achieve balance. This strategy can be viewed as *load balancing through load migration*. It is to perturb the original partitioning just enough each time to balance the load, which is much more efficient than repartitioning from scratch. Typically, the partitions produced by methods of this strategy are close to the old partitions and hence the amount of data needed to be redistributed is usually much smaller. Thus, this strategy is more practical for dynamic load balancing of DVEs. Our proposed load balancing algorithms belong to this strategy. It is based on a discrete approximation to a closed form solution of the continuous heat diffusion equation.

In this work, we investigate the effectiveness of applying the heat diffusion approach for load balancing in DVEs. There are two key contributions in this paper. First, we propose an *efficient cell selection scheme* to identify and select appropriate cells for migration among the neighbor servers, while maintaining the quality of the partition boundaries. Second, we present two heat diffusion based load balancing algorithms according to the level of information utilized in the decision making procedure, local and global diffusion. Our experiments show that both of them perform better than some existing algorithms, with the global method giving better overall performance.

The rest of this paper is organized as follows. Section 2 reviews the previous load migration based load balancing algorithms. Sec-

tion 3 presents our heat diffusion based load balancing algorithms. Section 4 shows and discusses some experimental results. Finally, Section 5 concludes this paper.

2 Related Work

There are a number of load balancing algorithms proposed that are based on load migration, e.g., [Lin and Keller 1987] [Cybenko 1989] [Boillat 1990] [Xu and Lau 1992] [Horton 1993] [Willebeek-LeMair and Reeves 1993] [Watts and Taylor 1998] [Ou and Ranka 1997] [Hu et al. 1998] [Hu and Blake 1999a] [Lui and Chan 2002] [Ng et al. 2002] [Lee and Lee 2003] [Gastner and Newman 2004] [Chen et al. 2005] [Ren et al. 2005]. In general, they can be classified into two main categories according to the level of knowledge that is embodied in the decision making process. The level of knowledge refers to the degree to which a server knows the status of other servers. Naturally, the more information that we have in the decision making process, the more accurate the resulting solution will be. However, as more information are exchanged among the servers, higher communication and computational costs will be incurred. By verifying the level of knowledge available for making the load balancing decision, existing algorithms can be classified into global algorithms and local algorithms. In general, a global algorithm receives the load information from all servers for the load balancing process, while a local algorithm receives information from only the neighbor servers.

2.1 Global Algorithms

There have been a lot of global algorithms proposed for different applications. Some of them are based on modeling the load migration problem as a global optimization problem. We refer to these algorithms as *global optimization based algorithms*. In [Hu et al. 1998], a global algorithm motivated by the need to minimize the amount of load migration for load balancing is proposed for the finite element method (FEM). In [Ou and Ranka 1997], the incremental graph partitioning problem is modeled as a linear programming problem with the objective of minimizing the amount of nodes for migration in the load balancing process. They use the simplex method to solve the linear programming problem. [Hu and Blake 1999b] points out that the method used to model the load migration problem in [Ou and Ranka 1997] is better than that in [Hu et al. 1998] as it not only considers the load imbalance but also the communication cost, which measures the edge-cut quality in graph partitioning.

In [Lui and Chan 2002], a global algorithm for DVEs is proposed, which is derived from [Ou and Ranka 1997]. It performs user migration so as to minimize a linear combination of the load imbalance among the servers and the inter-server communication cost of the resulting user redistribution. In each partitioning process, the algorithm constructs a global graph in which the nodes represent the users and each edge represents the communication between a pair of users. In order to form a connected graph, it may need to create extra edges that may not represent actual communication links among the users. In addition, there is no guarantee that there will be a solution for the linear optimization problem.

Although global optimization based algorithms can obtain good balancing results, they typically have high computational and communication costs, as they involve all nodes in the VE to perform the global optimization procedures. We note that in DVEs, it is more important to balance the load to an acceptable level as quickly as possible during runtime. Whether the load is ideally balanced or not is less critical. As a result, global optimization based algorithms are less suitable for DVEs.

2.2 Local Algorithms

In recent years, there are a few local algorithms proposed to address the load balancing problem of DVEs, including [Ng et al. 2002] [Lee and Lee 2003] [Chen et al. 2005]. In [Ng et al. 2002], an adaptive region partitioning algorithm is proposed to partition the VE into regions and each of them is handled by one server. When a server is overloaded, it will search for the lightest loaded neighboring server and transfer some load to this target server. This method is very efficient as it only needs to consider the loading status of the neighbor servers. However, it may not be able to disperse the load quickly if neighboring servers also have very high loading. In [Lee and Lee 2003], a revised method is proposed to address this limitation. Instead of finding a single target server, this method identifies a set of connected servers to the overloaded server and then performs optimized repartitioning on the relevant regions among these servers in order to achieve a better load balancing. However, this method has a higher computational cost compared with [Ng et al. 2002], due to the extra processing. [Chen et al. 2005] proposes another local load balancing algorithm, in which each server monitors its QoS violations measured in terms of the user response time, determines whether a perceived QoS violation is due to heavy workload or high inter-server communication, and then triggers either load shedding or load aggregation, respectively. This method considers not only to migrate load from the overloaded servers, but also to merge the regions with excessive inter-server communication. However, it also has a high computational cost.

Although these local load balancing algorithms can effectively resolve the large load imbalance, the balancing results usually tends to be short-term, as they only consider local server loading information to make the load balancing decisions. As a result, the load balancing process may have to be executed more frequently than the global load balancing algorithms, which typically produce better long-term results. Hence, a load balancing algorithm that would produce good balancing results, while taking up very little computation overhead is needed in DVEs.

2.3 Heat Diffusion Algorithms

The use of the heat diffusion process to guide the load migration has been a popular approach for dynamic load balancing in many applications, like multiprocessor computing, parallel FEM computing and parallel molecule dynamics simulation. Heat diffusion was first presented in [Cybenko 1989] for dynamic load balancing on message passing multiprocessor networks. The method can achieve global balance while requiring very little computation and communication overheads as it involves simple calculations and uses only local information. [Willebeek-LeMair and Reeves 1993] points out that the heat diffusion algorithm is superior to some global algorithms, like the gradient model algorithm [Lin and Keller 1987] and hierarchical algorithm [Horton 1993] in terms of quality and speed. In addition, [Hu and Blake 1998] proves that the heat diffusion algorithm has an optimal property of minimizing the amount of load needed to be migrated as compared with the global optimization based algorithms [Ou and Ranka 1997] [Hu et al. 1998]. The main problem of the heat diffusion algorithm as pointed out by [Boillat 1990] is that it may converge slowly in some graphs with very large scale and very small connectivity.

There have been some methods proposed to accelerate the convergence speed. [Watts and Taylor 1998] applies the second-order implicit scheme, instead of the first-order explicit scheme as in [Cybenko 1989], to shorten the convergence time. [Diekmann et al. 1997] [Muthukrishnan et al. 1998] [Hu and Blake 1999a] use the Chebyshev polynomial to do so. These algorithms can reduce the number of iterations for convergence. However, they require higher

computation and communication overheads.

Besides parallel computing applications, heat diffusion has also been employed to balance the bin density in VLSI physical design [Ren et al. 2005], and to balance the data density for cartograms in applied mathematics [Gastner and Newman 2004]. Although heat diffusion is popular and has proved to be very efficient for load balancing in many areas, to our knowledge, it has not been employed for load balancing in DVEs.

3 Heat Diffusion Based Load Balancing

We model the VE by dividing it regularly into a large number of square cells, $\{c_{i,j}\}_{n\times n}$. Each cell $c_{i,j}$ contains some objects, and its load is determined by the number of objects inside it. The VE is also partitioned into P regions, i.e., $R = \{r_1, r_2, ..., r_P\}$. Each region contains an integer number of cells and region r_i has a load of l_i. Hence, the corresponding load of R is: $L = \{l_1, l_2, ..., l_P\}$. If each region is assigned to a separate server, there should be P servers in the system, $S = \{s_1, s_2, ..., s_P\}$, with s_i serving region r_i of load l_i. Figure 1 illustrates the VE.

Based on the state of the VE, we construct a server graph $G = (V, E)$, where each node $v_i \in V$ represents server s_i (managing region r_i) with load l_i. Each edge $e_{ij} \in E$ connects nodes v_i and v_j. Figure 2 shows the server graph of Figure 1. The load balancing problem is then to determine the amount of load to migrate along each edge with the objective of equalizing the load in each server.

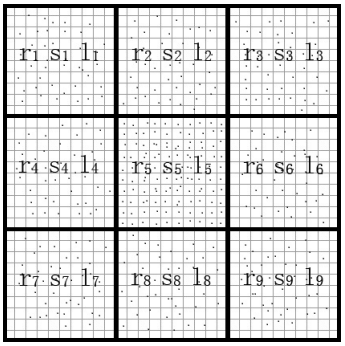


Figure 1: *A VE is subdivided into* 30×30 *cells and 9 regions (in thick lines), with objects shown as dots.*

The main idea of the heat diffusion algorithm is that given a server graph, each node sends some load to its adjacent nodes in each iteration. The amount to transfer to an adjacent node is proportional to the difference between the load of the current node and that of the adjacent node. This process may take several iterations in order to roughly balance the load of each node. This load migration process is analogous to the heat diffusion process, which is governed by a parabolic partial differential equation (or the heat equation):

$$\frac{\partial u}{\partial t} = \alpha(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{1}$$

where $u(x, y)$ describes the temperature at a given spatial location (x, y), t is the time, α is the diffusivity, and $\partial u/\partial t$ is the rate of temperature change at (x, y). $\partial^2 u/\partial x^2$ and $\partial^2 u/\partial y^2$ are the second spatial derivatives (thermal conductions) of temperature in the x and y directions, respectively. The heat equation is used to determine the change of the temperature distribution over time. Solutions of the heat equation are characterized by a gradual smoothing of the initial temperature distribution by the flow of heat from warmer to colder areas of an object. In DVEs, we may employ a discrete form of Eq. (1) by the first-order explicit scheme [Cybenko 1989] for dynamic load balancing.

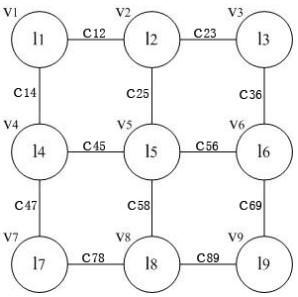


Figure 2: *The server graph corresponding to the VE in Figure 1, with the diffusion coefficient marked on each edge.*

In server graph $G = (V, E)$, edge e_{ij} represents the channel between two nodes v_i and v_j. v_i may send load to (or receive load from) v_j through e_{ij}. The load migration from v_i to v_j can be represented by the potential flow $f_{i\to j}$. To compute the potential flow through each edge, we let the load of v_i be $l_i^{(0)}$ initially and $l_i^{(k)}$ at iteration k. According to [Cybenko 1989], the load $l_i^{(k+1)}$ at iteration $k + 1$ is:

$$l_i^{k+1} = l_i^k - \sum_j c_{ij}(l_i^k - l_j^k) \tag{2}$$

where c_{ij} is the diffusion coefficient associated with edge e_{ij} and should be chosen properly in order to achieve global balance. [Boillat 1990] suggests a way to determine the diffusion coefficients for general connected graphs as follows:

$$c_{ij} = \frac{1}{max\{deg(v_i), deg(v_j)\} + 1} \tag{3}$$

where $deg(v_i)$ is the degree of node v_i, i.e., the number of edges connected to it. We apply Eq. (3) in our algorithms for choosing the diffusion coefficients because the server graph of a DVE can be any kind of connected graphs.

According to the level of knowledge which is embodied in the load balancing decision making process, we present two load balancing algorithms for DVEs based on heat diffusion. One is the *local diffusion algorithm* in which the load balancing decision making process utilizes only local load information and is carried out by every server. The other is the *global diffusion algorithm* in which the load balancing decision making process utilizes global load information and is carried out by a single server (or the central server).

3.1 The Local Diffusion Algorithm

In a DVE, some regions may be more attractive than others due to some application dependent reasons. So, more users may concentrate in these regions. Servers serving these regions will take up higher loads than others, which causes load imbalance. The *local diffusion algorithm* aims to improve the load balancing problem in a way that if a server has more load than some of its neighbor servers, it transfers (or diffuses) some of its load to those servers.

3.1.1 Local Decision Making

Let server s_i have a neighbor server s_j such that edge e_{ij} exists. From Eq. (2), we may compute the amount of load to be transferred from s_i to s_j, which we refer to as a potential flow $f_{i\to j}$:

$$f_{i\to j} = c_{ij}(l_i - l_j) \tag{4}$$

At each load balancing process, each server s_i will determine the potential flow $f_{i\rightarrow j}$ to each of its neighbor servers s_j. Theoretically, after a number of load balancing processes, the load imbalance will be diffused among all the servers and the loads of all the servers will become balanced. However, in practice, the actual load migration process is considerably application specific, due to the differences in the definition of load in different applications. In our DVE, we subdivide the virtual environment into cells and each cell is considered as a basic unit for transfer among the servers. On the other hand, the load in the virtual environment is measured in terms of the number of users. As each cell may have a different number of users, i.e., a different load, there is not a direct mapping from the amount of load to transfer into the number of cells to transfer. To address this problem, we propose an *efficient cell selection scheme* to carry out the load transfer effectively.

3.1.2 Efficient Cell Selection Scheme

Our cell selection scheme can be divided into two processes: an initial process and a runtime process. During the initial process, for each node v_i of the server graph, we construct a list of boundary cells with each of v_i's neighbor nodes, v_j. Each of these boundary cells has at least one side connected to v_j. We refer to this list as the *boundary cell list*. (Hence, if v_i has 4 neighbor nodes, it should maintain four boundary cell lists.) We then assign a priority to each cell of this boundary cell list according to the number of sides that it has connected to v_j. As there is at least one side and at most three sides of each cell connected to v_j, the priority values range between 1 to 3. Finally, we sort this list in descending order of the priority value.

During the runtime process, when server v_i obtains the amount of load, $f_{i\rightarrow j}$, to transfer to v_j, if it is a negative value, it means that v_i will receive some load from v_j and will not need to carry out the cell selection scheme. If it is a positive value, we first round it up to the nearest integer number of users to be transferred, $\lceil f_{i\rightarrow j} \rceil$. We then perform the following operations:

(1) Select the first cell in the list for transfer.

(2) Check and update the status of the four neighbor cells of the transferred cell. This may involve adding them to the appropriate boundary cell lists and updating their priority values.

(3) Reduce $\lceil f_{i\rightarrow j} \rceil$ by the load in the transferred cell.

(4) Repeat step 1 if $\lceil f_{i\rightarrow j} \rceil > 0$. Otherwise, we have finished with e_{ij}.

3.2 The Global Diffusion Algorithm

Our global diffusion algorithm has two key stages: *global scheduling stage* and *local load migration stage*. In the global scheduling stage, the central server, which has the global knowledge of all servers, computes the amount of load for each node to transfer to each of its adjacent nodes through the corresponding edge. This stage typically involves multiple iterations of potential flow computation. In the load migration stage, each server carries out the load transfer locally.

3.2.1 Global Scheduling Stage

At this stage, a central server will compute the amount of load to be migrated through each edge in the server graph based on the load information of all the servers.

In Eq. (2), $c_{ij}(l_i^k - l_j^k)$ is the potential amount of load to be migrated through edge e_{ij} at iteration $k+1$. According to [Cybenko 1989], when the diffusion algorithm has converged, the load distribution would reach a uniform distribution, $\bar{L} = \{\bar{l}, \bar{l}, ...\bar{l}\}$, where $\bar{l}$ is the average load of all the servers. As we have mentioned, we do not really need to achieve such an exact load balance in DVEs. Besides, our loads cannot be unlimitedly divided. Supposed that we need a total of n iterations in order to reach a roughly load balance. We will then obtain the accumulated potential flow through each edge e_{ij} as:

$$F_{i\rightarrow j} = \sum_{k=0}^{n-1} c_{ij}(l_i^k - l_j^k) \tag{5}$$

where $F_{i\rightarrow j}$ indicates the total amount of load that node v_i should send to (if $F_{i\rightarrow j} > 0$) or receive from (if $F_{i\rightarrow j} < 0$) its neighbor node v_j.

After finishing the global scheduling stage, the central server will inform each server in the server graph the accumulated potential flow for each of its adjacent edges.

3.2.2 Local Load Migration Stage

If the communication delay between the central server and each of the servers in the server graph can be negligible, the central server may run the global load balancing process immediately after receiving the current load information from all the servers, and returns the accumulated potential flows to all the servers just before the servers start the next frame. Upon receiving this potential flow information, i.e., the set of $F_{i\rightarrow j}$ values stated in Eq. (5), each server will simply execute the efficient cell selection scheme as described in Section 3.1.2 by replacing $f_{i\rightarrow j}$ with $F_{i\rightarrow j}$.

However, in a DVE where the servers may be distributed over the networks, some servers may have lower latency values to the central servers while other may have higher ones. This latency between the central server and the other servers causes two major problems. First, as there can be a significant time gap between the moment when a server sends its load information to the central server and the moment when the server receives the set of $F_{i\rightarrow j}$ values from the central server, the latest load of the server can be very different from the one that the central server used to determine the accumulated potential flow. Second, when the latency values are comparable to the frequency of executing the load balancing process, the previous local load migration stage may not have been completed before the central server enters a new global scheduling stage. In other words, the central server is using the old load information for the new global scheduling stage. This will cause excessive amount of load to be transferred among the servers, resulting in significant load fluctuation.

To address the first problem, we propose a simple method to adjust the accumulated potential flow. We assume that at time t_1, server v_i sends its current load value, $l_i(t_1)$, to the central server, which then computes the set of $F_{i\rightarrow j}$ values for all servers. When v_i receives its own set of $F_{i\rightarrow j}$ values at time t_2, its load becomes $l_i(t_2)$. Instead of directly using $F_{i\rightarrow j}$ in the efficient cell selection scheme, we adjust its value in the following way:

$$F'_{i\rightarrow j} = \frac{l_i(t_2)}{l_i(t_1)} F_{i\rightarrow j} \tag{6}$$

We then replace $f_{i\rightarrow j}$ with $F'_{i\rightarrow j}$ in the efficient cell selection scheme to select cells for migration. Our experiments show that this produces a reasonable good approximation.

To address the second problem, we make sure that the duration between two consecutive load balancing processes, t_{clb}, is longer than the duration of a complete load balancing process. Hence, we need to satisfy the following condition:

$$t_{clb} > (\mathrm{MAX}_{v_i \in V}(t_{rt,i}) + t_{lb}) \tag{7}$$

where $t_{rt,i}$ is the round trip delay between server v_i and the central server. MAX() finds the maximum round trip delay from all the servers in the DVE. t_{lb} is the time taken by the central server to run the global scheduling procedure.

4 Results and Discussions

To evaluate the performance of the proposed methods, we have implemented a simulation environment of the DVE in C++. Our testing platform is a PC with an Intel Core Duo 2.4GHz CPU and 2GB RAM. We have created a VE of 600 $\times$ 600 m^2 in size. It is divided into 100 regions managed by 100 servers. The load of a region is determined by the number of users within it. We set the maximum load of a server to be 50 users. Hence, a server will be considered as overloaded when the number of users within its region reaches 50.

We have simulated two types of users, individuals and crowds. Individual users move in random fashion, changing their speeds and directions randomly at each step. Crowds are clusters of users that move together according to some specified paths. The introduction of large crowds moving around the VE is to test how the load balancing algorithms respond in such a demanding situation. We have created a total of 4,000 users and simulated two scenarios:

- Scenario 1 - Demanding scenario:

  3,000 individual users move around the VE at speeds randomly changing from 0 to 0.5m per frame. In addition, a crowd of 1,000 users is set to move along the boundary of the VE at a randomly changing speed from 0 to 0.5m per frame.

- Scenario 2 - Highly-demanding scenario:

  2,000 individual users are moving around the VE at speeds randomly changing from 0 to 0.5m per frame. A crowd of 1,000 users is set to move along the boundary of the VE at a randomly changing speed from 0 to 0.5m per frame. Another crowd of 1,000 users is set to move behind the first crowd at twice the speed of the first crowd. This scenario is more demanding than Scenario 1 as it has two crowds. The most demanding moment is when the two crowds meet, causing a high concentration of users within a small area.

The system frame rate is set to 10. Hence, the system updates the location of each user in the VE every 0.1s. In our experiments, we ran each scenario for 300 seconds. Figure 3 shows a snapshot of the simulated virtual environment, with each color region representing one of the 100 regions. The numbers within each region show the corresponding server ID and load value. A crowd of 1,000 users is currently located around the center of the virtual environment. The three regions with white borders are overloaded servers.

To evaluate our load balancing algorithms, we introduce two performance metrics here:

- **Number of Overloaded Servers** (N_{OS}):

  This counts the average number of servers getting overloaded within a certain period of time (as proposed in [Lau 2010]). Here, we add up the total number of overloaded servers over 1 second. (If a server is overloaded in 3 frames out of a second, it is counted as 3 overloaded servers.) We then divide this number by the frame rate, i.e., 10, to obtain the N_{OS} value. This metric is to evaluate the ability of the load balancing algorithm in avoiding server overloading. A good load balancing algorithm should have a low N_{OS} value.

- **Number of Migrated Users** (N_{MU}):

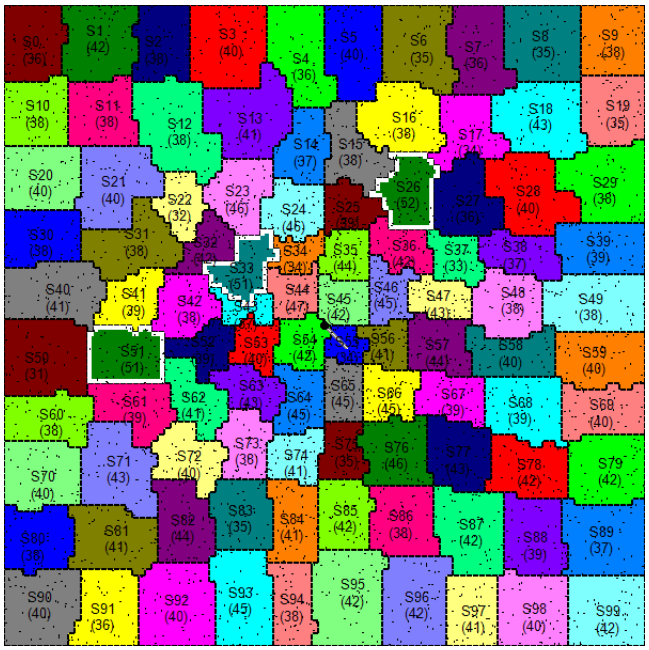


Figure 3: *A snapshot of our simulated DVE with 100 regions served by 100 servers.*

  This counts the number of users being migrated within a certain period of time. Migrating a large number of users consumes network bandwidth. This metric is to evaluate the transmission overhead of the load balancing algorithm. A good load balancing algorithm should have a low N_{MU} value.

4.1 Experiment 1

In this experiment, we compare the performance of the two proposed load balancing algorithms with the adaptive region partitioning algorithm proposed in [Ng et al. 2002]. To simplify the discussion, we refer to our local diffusion algorithm as **LD**, our global diffusion algorithm as **GD**, and the adaptive region partitioning algorithm as **Adaptive**. As we do not consider network latency in this experiment, we execute the three load balancing algorithms in every frame.

We use the two performance metrics, N_{OS} and N_{MU}, to compare the performance of the three methods. Figures 4(a) and 4(b) show their performances under scenario 1, with 1 crowd moving around the VE. Figures 4(c) and 4(d) show their performances under scenario 2, with 2 crowds moving around the VE. We can clearly see that both **LD** and **GD** generated much lower numbers of overloaded servers and required to migrate much smaller numbers of users throughout the simulation. The reason is that both methods try to diffuse any load differences to the neighbor servers as soon as the differences arise. On the other hand, **Adaptive** tries to reduce the load only when a server is overloaded. Unfortunately, very often when a server is overloaded, its neighbor servers may also be overloaded or nearly overloaded. This causes the loads to be passing around the overloaded or nearly overloaded servers, resulting in more servers getting overloaded and more users needed to be migrated.

In fact, we have tried lowering the threshold for triggering the load balancing process in **Adaptive** to 90%. Whenever a server's load reached 90% of its maximum load, it would start the load balancing process. We thought that this might reduce the number of server

overloading. To our surprise, N_{OS} was roughly the same as those shown in Figures 4(a) and 4(c). However, N_{MU} went up significantly to about 3 times of the numbers shown in Figures 4(b) and 4(d). Our explanation is that since we have a total of 4,000 users and 100 servers, the average number of users per server is 40. The maximum number of users that a server can serve is set to 50. If we reduce the threshold to 90%, a server will start the load balancing process whenever its number of users reaches 45. This results in a small margin of 5 users and may cause more users to be migrated unnecessarily.

Comparing between **LD** and **GD**, we can see that both methods needed to migrate very similar numbers of users. However, **GD** generated smaller numbers of overloaded servers on average for both scenarios. This is because **GD** used the global load information to compute accumulated flows for each server, which produces better overall performance.

4.2 Experiment 2

In this experiment, we study the computational costs of the three methods. Table 1 shows the average computation times of the three methods under the two scenarios. Note that the settings were the same as in Experiment 1. The way that we determined the computation times was by identifying the server that took longest to run the load balancing process in each frame. We then added up all these maximum times over all the frames and average them by the total number of frames. The computation times for **GD** include both the global scheduling stage and the local load migration stage.

	Adaptive	**LD**	**GD**
Scenario 1	7.11 ms	2.01 ms	2.96 ms
Scenario 2	26.75 ms	2.49 ms	3.46 ms

Table 1: *Computational costs of the three algorithms.*

We can see that both **LD** and **GD** are far more efficient than **Adaptive**. This is mainly due to the simplicity of the diffusion algorithms used in the two methods. **GD** is slightly more expensive than **LD** due to the need to process the global load information.

4.3 Experiment 3

In this experiment, we study how the network delay affects the performance of **GD**. We have considered three situations of network delay: *negligible delay*, *1 to 2 frames delay*, and *2 to 3 frames delay*. The first situation simulates the LAN environment, where the network delay is negligible. The other two situations simulate real distributed environments, where the delay can vary among the servers. The second situation models a network environment with a maximum of 2-frame delay (i.e., 0.2s) and the third situation models a network environment with a maximum of 3-frame delay (i.e., 0.3s). To satisfy the condition stated in Eq. (7), we set the frequency of running **GD** as once per second.

Figure 5 shows the performance of **GD** in the three situations under scenario 2. We can see from Figure 5(b) that the numbers of migrated users in the three situations are very similar. So, N_{MU} is not really affected by the network delay here. On the other hand, we can see from Figure 5(a) that there is a slight increase in the number of overloaded servers as we increase the network delay. This is mainly due to the fact that as the network delay increases, it will take longer for **GD** to react to the server overloading.

It is also interesting to compare the N_{OS} curve of **GD** with negligible delay in Figure 5(a) with that of **GD** in Figure 4(c). Both methods assume no network delay. The only difference is their frequency of running **GD**. We can see that as we reduce the frequency from once per frame (as shown in Figure 4(c)) to once per second (as shown in Figure 5(a)), the average number of overloaded servers increases from about 0.1 to about 1. Hence, we should maximize this frequency.

5 Conclusion

We have proposed a new dynamic load balancing approach for DVEs based on heat diffusion, which has been studied in other areas and proved to be a very effective and efficient tool for dynamic load balancing. We have developed an *efficient cell selection scheme* to identify and select appropriate cells for load migration under the heat diffusion framework in DVEs. We have investigated two heat diffusion based load balancing algorithms, local diffusion and global diffusion. Our experiments show that the global diffusion method performs better than the local one and much better than our earlier adaptive region partitioning algorithm. To our knowledge, this is the first time that the heat diffusion approach has been used for load balancing in DVEs.

One possible limitation of our global diffusion algorithm is that it may suffer from significant network delay when the servers are distributed across networks. In this paper, we only briefly address some of the relevant issues. As a future work, we are currently conducting a more detailed investigation into the relevant problems.

Acknowledgements

The work described in this paper was partially supported by a GRF grant from the Research Grants Council of Hong Kong (RGC Reference Number: CityU 116008) and a SRG grant from City University of Hong Kong (Project Number: 7002463).

References

BERGER, M., AND BOKHARI, S. 1987. A Partitioning Strategy for Nonuniform Problems on Multiprocessors. *IEEE Trans. on Computers 36*, 5, 570–580.

BOILLAT, J. 1990. Load Balancing and Poisson Equation in a Graph. *Concurrency: Practice and Experience 2*, 4, 289–313.

BUI, T., AND MOON, B. 1996. Genetic Algorithm and Graph Partitioning. *IEEE Trans. on Computers 45*, 7, 841–855.

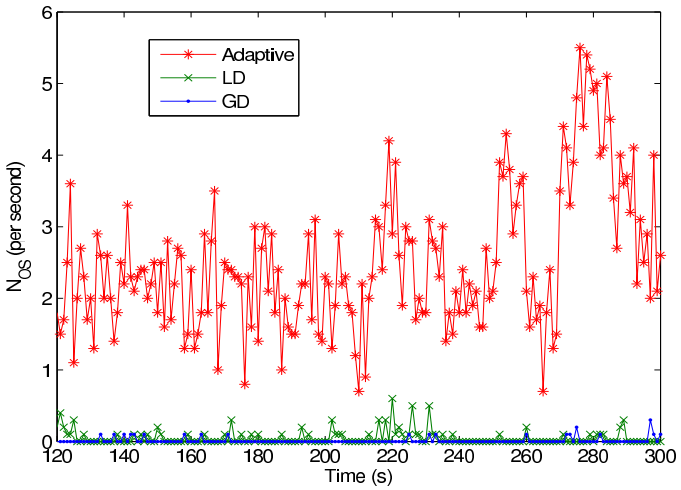
CHEN, J., WU, B., DELAP, M., KNUTSSON, B., LU, H., AND AMZA, C. 2005. Locality Aware Dynamic Load Management for Massively Multiplayer Games. In *Proc. ACM SIGPLAN Symp. on PPoPP*, 289–300.

CYBENKO, G. 1989. Dynamic Load Balancing for Distributed Memory Multiprocessors. *Journal of Parallel and Distributed Computing 7*, 2, 279–301.

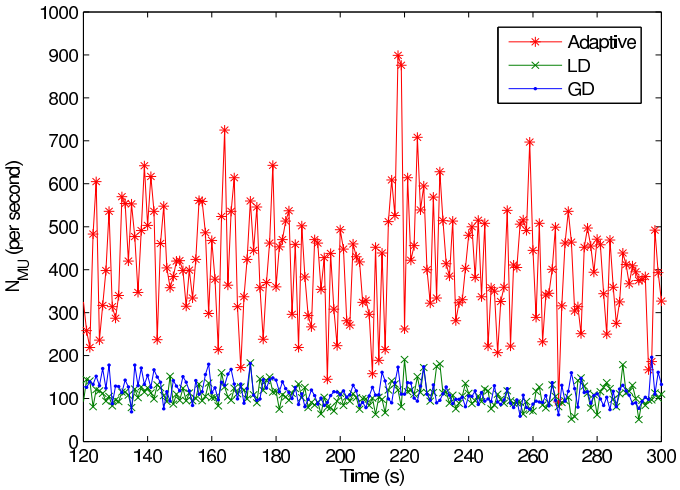
DIEKMANN, R., MUTHUKRISHNAN, S., AND NAYAKKANKUPPAM, M. 1997. Engineering Diffusive Load Balancing Algorithms Using Experiments. In *Proc. International Symp. on Solving Irregularly Structured Problems in Parallel*, 111–122.

FUNKHOUSER, T. 1995. RING: A Client-Server System for Multi-User Virtual Environments. In *Proc. ACM I3D*, 85–92.

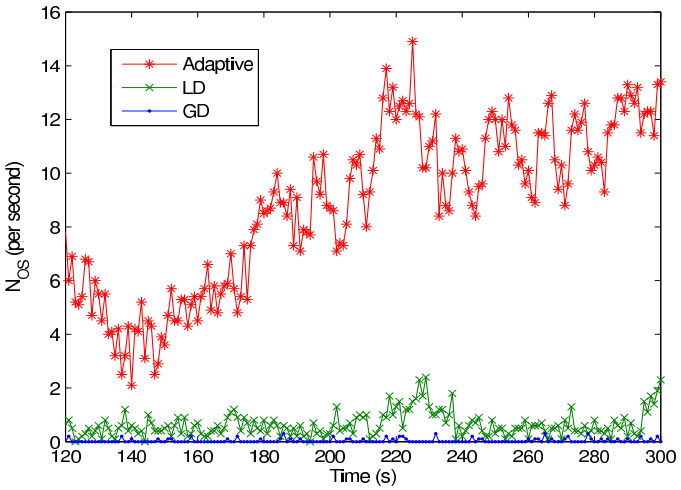
GASTNER, M., AND NEWMAN, M. 2004. Diffusion-based Method for Producing Density-equalizing Maps. *Proc. National Academy of Sciences 101*, 20, 7499–7504.



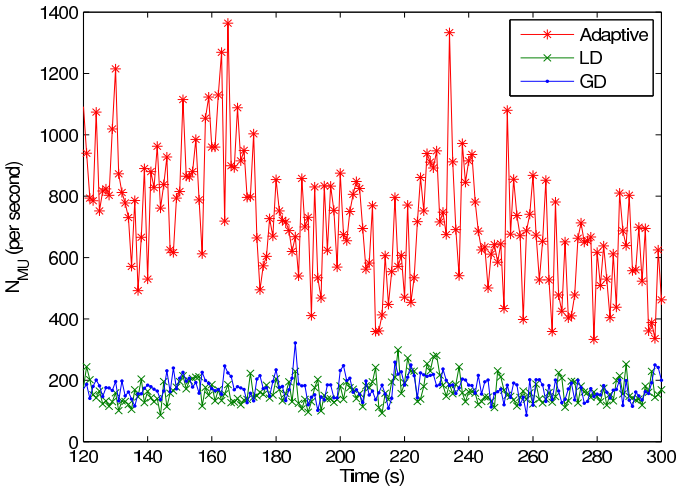
(a) N_{OS} *for Scenario 1*



(b) N_{MU} *for Scenario 1*

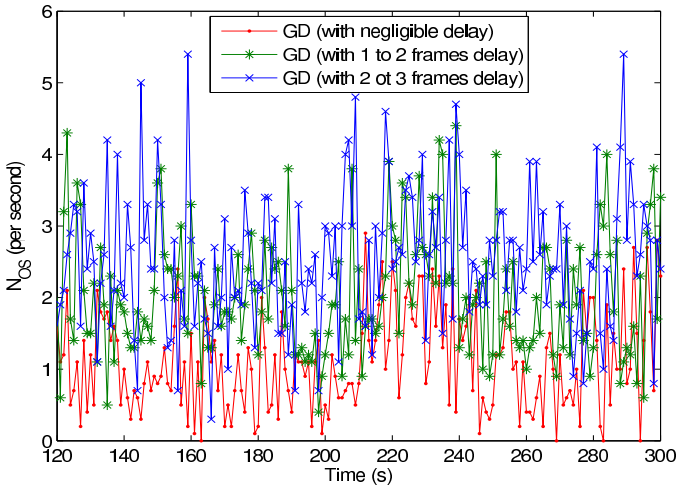


(c) N_{OS} *for Scenario 2*

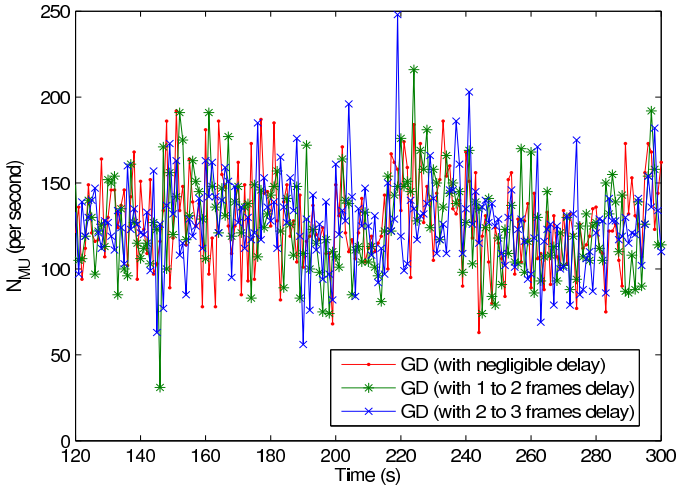


(d) N_{MU} *for Scenario 2*

Figure 4: *Comparison of the performance among various load balancing algorithms in Experiment 1.*



(a) N_{OS} *for Scenario2*



(b) N_{MU} *for Scenario2*

Figure 5: *Performance of GD under various network delays in Experiment 2.*

HORTON, G. 1993. A Multi-level Diffusion Method for Dynamic Load Balancing. *Parallel Computing 19*, 2.

HU, Y., AND BLAKE, R. 1998. The Optimal Property of Polynomial based Diffusion-like Algorithms in Dynamic Load Balancing. *Computational Dynamics 98*, 177–183.

HU, Y., AND BLAKE, R. 1999. An Improved Diffusion Algorithm for Dynamic Load Balancing. *Parallel Computing 25*, 4, 417–444.

HU, Y., AND BLAKE, R. 1999. Load Balancing for Unstructured Mesh Applications. *Parallel and Distributed Computing Practices 2*, 3, 117–148.

HU, Y., BLAKE, R., AND EMERSON, D. 1998. An Optimal Migration Algorithm for Dynamic Load Balancing. *Concurrency: Practice and Experience 10*, 6, 467–483.

LAU, R. 2010. Hybrid Load Balancing for Online Games. In *Proc. ACM Multimedia (to appear)*.

LEE, K., AND LEE, D. 2003. A Scalable Dynamic Load Distribution Scheme for Multi-server Distributed Virtual Environment systems with highly-skewed user distribution. In *Proc. ACM VRST*, 160–168.

LIN, F., AND KELLER, R. 1987. The Gradient Model Load Balancing Method. *IEEE Trans. on Software Engineering 13*, 1, 32–38.

LUI, J., AND CHAN, M. 2002. An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems. *IEEE Trans. on Parallel and Distributed Systems 13*, 3, 193–211.

MORISON, R., AND OTTO, S. 1987. The Scattered Decomposition for Finite Elements. *Journal of Scientific Computing 2*, 1, 59–76.

MUTHUKRISHNAN, S., GHOSH, B., AND SCHULTZ, M. 1998. First- and Second-order Diffusive Methods for Rapid, Coarse, Distributed Load Balancing. *Theory of Computing Systems 31*, 4, 331–354.

NG, B., SI, A., LAU, R., AND LI, F. 2002. A Multi-server Architecture for Distributed Virtual Walkthrough. In *Proc. ACM VRST*, 163–170.

OU, C., AND RANKA, S. 1997. Parallel Incremental Graph Partitioning. *IEEE Trans. on Parallel and Distributed Systems 8*, 8, 884–896.

OU, C., RANKA, S., AND FOX, G. 1996. Fast and Parallel Mapping Algorithms for Irregular Problems. *The Journal of Supercomputing 10*, 2, 119–140.

REN, H., PAN, D., ALPERT, C., AND VILLARRUBIA, P. 2005. Diffusion-based Placement Migration. In *Proc. Design Automation Conference*, 515–520.

SIMON, H. 1991. Partitioning of Unstructured Problems for Parallel Processing. *Computing Systems in Engineering 2*, 2-3, 135–148.

WATTS, J., AND TAYLOR, S. 1998. A Practical Approach to Dynamic Load Balancing. *IEEE Trans. on Parallel and Distributed Systems 9*, 3, 235–248.

WILLEBEEK-LEMAIR, M., AND REEVES, A. 1993. Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans. on Parallel and Distributed Systems 4*, 979–993.

WILLIAMS, R. 1991. Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations. *Concurrency: Practice and Experience 3*, 5, 457–481.

XU, C., AND LAU, F. 1992. Analysis of the Generalized Dimension Exchange Method for Dynamic Load Balancing. *Journal of Parallel and Distributed Computing 16*, 4, 385–393.