

Dynamic Load Balancing in Distributed Virtual Environments using Heat Diffusion

YUNHUA DENG and RYNSON W.H. LAU, City University of Hong Kong

Distributed virtual environments (DVEs) are attracting a lot of attention in recent years, due to the increasing popularity of online gaming and social networks. As the number of concurrent users of a DVE increases, a critical problem is on how the workload among multiple servers can be balanced in order to maintain real-time performance. Although a number of load balancing methods have been proposed, they either try to produce high quality load balancing results and become too slow or emphasize on efficiency and the load balancing results become less effective. In this paper, we propose a new approach to address this problem based on heat diffusion. Our work has two main contributions. First, we propose a local and a global load balancing methods for DVEs based on heat diffusion. Second, we investigate two performance factors of the proposed methods, the convergence threshold and the load balancing interval. We have conducted a number of experiments to extensively evaluate the performance of the proposed methods. Our experimental results show that the proposed methods outperform existing methods in that our methods are effective in reducing server overloading while at the same time being efficient.

Categories and Subject Descriptors: C.2.4 [**Distributed Systems**] Client/server; H.5.1 [**Multimedia Information Systems**] Artificial, augmented, and virtual realities; I.3.2 [**Graphics Systems**] Distributed/network graphics; I.3.7 [**Three-Dimensional Graphics and Realism**] Virtual reality

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Multi-server architecture, DVE load balancing, heat diffusion, network latency.

ACM Reference Format:

Y. Deng and R. Lau 2013. Dynamic Load Balancing in Distributed Virtual Environments using Heat Diffusion. ACM Trans. Multimedia Comput. Commun. Appl. 0, 0, Article 0 (2013), 20 pages.
DOI = 10.1145/0000000.0000000 <http://doi.acm.org/10.1145/0000000.0000000>

1. INTRODUCTION

A distributed virtual environment (DVE) allows remote users to interact in a 3D virtual environment (VE) through the Internet. Applications of DVEs include multiplayer online games, military/industrial remote training, collaborative engineering and social networks. As the scale of some DVE systems is becoming very large with thousands of concurrent users, the DVE load balancing problem has been attracting a lot of research attention.

Author's address: Y. Deng, Department of Computer Science, City University of Hong Kong, Kowloon Tong, Kowloon, Hong Kong; email: yunhua.deng@my.cityu.edu.hk; R. Lau, Department of Computer Science, City University of Hong Kong, Kowloon Tong, Kowloon, Hong Kong; email: rynson.lau@cityu.edu.hk

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2013 ACM 1551-6857/2013/-ART0 \$10.00

DOI 10.1145/0000000.0000000 <http://doi.acm.org/10.1145/0000000.0000000>

A popular way to address the DVE load balancing problem is to divide the load among multiple servers by dividing the VE into multiple partitions with each partition being served by a single server. Some load balancing methods may treat this partitioning process as an offline process. This load balancing strategy is referred to as *static load balancing*. For example, [Steed and Abou-Haidar 2003] statically divide the VE into multiple partitions according to the expected density of users and number of handovers. They consider several partitioning methods, such as quad-tree, unconstrained and constrained K-D tree, and region growing. Similar study is also conducted by [Prasetya and Wu 2008], where a set of partitioning methods with different partition shapes are compared. Both studies assume the VE to be partitioned in advance. Some popular DVE applications, such as World of Warcraft and Second Life, have adopted the static load balancing strategy. Its advantages are that it is simple and the speed of the partitioning process does not affect the interactivity of the DVE systems. However, as users of a DVE system move inside the VE, some regions may become too crowded while others may have too few users. As a result, the servers managing the crowded regions will become overloaded and the users within them may suffer from significant delay, while other servers may be under-utilized. (In systems like World of Warcraft and Second Life, players of different partitions are not allowed to interact with each other to avoid the synchronization problem. Unfortunately, this also reduces the shared experience of users.) To overcome this limitation, the partitioning process can be performed dynamically during runtime. This load balancing strategy is referred to as *dynamic load balancing*. The idea is that when the load of a server becomes too high, it may dynamically transfer part of its load to other lightly loaded servers to achieve load balancing.

There are two main approaches in dynamic load balancing, *global load balancing* and *local load balancing*, categorized by the level of knowledge utilized in the load balancing process. In global load balancing, a central process, possibly running on a central server, is responsible for computing the load balancing solutions for all servers based on the load information collected from all servers. This approach typically emphasizes on the accuracy of the load balancing solutions and neglects the computation overhead. Unlike global load balancing, local load balancing does not have a central process. Every server is responsible for computing its own load balancing solutions based on the load information of its nearby servers. As each server uses only the load information of nearby servers in the load balancing process, the load balancing process tends to be efficient but the load balancing solutions produced are often not very accurate.

In this work, we aim at addressing the limitations of the global and the local load balancing approaches. The major limitation of existing global approaches is of inefficient, which is mainly due to their high complexity, while the major limitation of existing local approaches is of ineffective, which is mainly due to the limited knowledge available to the load balancing process. Our objective of this work is to develop methods that are both effective and efficient in resolving DVE load balancing problems dynamically. To achieve this objective, we consider the use of heat diffusion approach here because of its two main advantages. First, heat diffusion has a simple concept while producing effective load balancing solutions [Cybenko 1989]. Second, it has a low computational cost since it involves only simple computations, without the need to solve any optimization [Willebeek-LeMair and Reeves 1993]. Hence, in this paper ¹, we investigate the effectiveness of applying the heat diffusion concept on DVE load balancing. Our main contributions include:

- (1) We investigate two methods based on heat diffusion, *local diffusion* and *global diffusion*, to address the dynamic load balancing problem of DVEs. As shown in our experiments, these two methods

¹This paper is an extension of our earlier conference paper in [Deng and Lau 2010].

are effective in balancing the load among servers at runtime, while having very low computation overheads. We also propose an *efficient cell selection scheme* to carry out the actual load transfer.

- (2) We investigate two performance factors, the convergence threshold and the load balancing interval. The convergence threshold determines when to stop the iteration in the global diffusion method, while the load balancing interval determines the shortest interval that we may perform the load balancing process in the global and the local diffusion methods, in the presence of network latency.

This is the first attempt to employ heat diffusion to solve the dynamic load balancing problem of DVEs. The rest of this paper is organized as follows. Section 2 summarizes related works. Section 3 presents our two heat diffusion based methods for DVE load balancing. Section 4 presents and discusses a number of experiments.

2. RELATED WORK

In this section, we summarize existing methods that are based on the global and on the local load balancing approaches. We also briefly discuss a hybrid load balancing method proposed recently. Finally, we summarize major diffusion methods proposed in other areas.

2.1 Global Load Balancing

Global load balancing methods utilize the load information from all servers to compute load balancing solutions, with the goal of keeping the load on every server balanced during runtime. In [Ou and Ranka 1997] and [Hu et al. 1998], the global load balancing problem is modeled as an optimization problem with the objective of minimizing the amount of load for migration during the load balancing process. Although these works are conducted primarily by researchers working in the area of finite element analysis, they address similar partitioning problems as ours.

Based on [Ou and Ranka 1997], [Lui and Chan 2002] proposes a global load balancing method for DVEs. It models the load of a DVE system as a connected graph, with each node representing a user and each edge representing the communication cost between the two adjacent nodes. It then performs node migrations between adjacent partitions so as to minimize a linear combination of the load imbalance among partitions and the inter-partition communication cost of the reassigned node distribution. However, in order to form a connected graph, this method may need to create extra edges that do not represent actual communication links among the users. Besides, as its optimization process involves all the nodes in the DVE system, it has a very high computational cost. (For example, it took about 200 and 800 seconds with 8 and 16 servers, respectively, as reported by the paper.) In [Van Den Bossche et al. 2009], the entire VE is divided into a set of cells. The optimal assignment of cells to servers with the objective of balancing load among servers is formulated as an integer linear programming problem. This problem is NP-hard, so that only suboptimal solutions can be obtained by adding more constraints including limiting the allowed computation time for the solver. As reported in the paper, the allowed computation time was set to 300 seconds with no more than 8 servers, but it may still be too short to obtain good solutions.

In [Ta et al. 2009], the zone mapping problem that concerns how to assign zones of the VE to a number of distributed servers with the objective of minimizing network latency between clients and servers is studied. This is an important problem especially for large-scale DVEs with lots of distributed servers and remote users. The method tries to avoid assigning a user to a distant server (in terms of network latency), which may degrade the interactivity of the DVE system as perceived by the user. [Ta et al. 2011] extends [Ta et al. 2009] by considering not only network latency between clients and servers, but also the re-mapping overhead (zone migrations) as the optimization objectives and use some heuristic algorithms to solve such a multi-objective optimization problem in a global manner.

However, this zone mapping problem (with the key objective of reducing the network latency between users and servers) is different from the load balancing problem (with the key objective of balancing the load among servers) that we address in this paper.

In summary, global load balancing methods generally produce effective load balancing solutions. However, they need a central server to process all the nodes, i.e., users, in an optimization framework. Hence, they have high computational costs and may not be practical for interactive DVE applications.

2.2 Local Load Balancing

Unlike global load balancing methods, which focus mainly on the quality of load balancing solutions, local load balancing methods focus more on the efficiency of the load balancing process. Each server in a local load balancing method needs to determine the amount of load to be transferred to its neighbor servers that manage the adjacent partitions.

In [Ng et al. 2002], an adaptive partitioning method is proposed. When a server is overloaded, it identifies the lightest loaded neighbor server as the target server to transfer its extra load to. It also attempts to minimize the perimeter length of the resulting partition as it determines which parts of the partition to transfer. This helps reduce the number of users located near the partition boundaries, i.e., the number of users needed to be served by two adjacent servers. Results show that this method is very efficient, due to its simplicity. However, as it only considers the load status of adjacent servers, the load balancing solutions produced tend to be short-term and the overloaded servers may quickly become overloaded again. In addition, an overloaded server may sometimes not be able to find any target servers to migrate its load if all its neighbor servers are also overloaded or highly loaded. In [Lee and Lee 2003], a refined method is proposed to address this limitation. When an overloaded server finds out that it cannot migrate all its extra load to its direct neighbor servers, it will consider the neighbor servers of its neighbor servers and so on. When there are enough target servers to take up all the excessive load, the partitions managed by all these selected target servers will be repartitioned using a graph partitioning method similar to [Lui and Chan 2002]. This method generally produces better load balancing solutions than [Ng et al. 2002] at the cost of a higher computation overhead.

In [Chen et al. 2005], each server monitors its own QoS violations measured in terms of user response time. It determines whether a perceived QoS violation is due to heavy workload, i.e., the server being overloaded, or high inter-server communication, i.e., the partition containing too many disconnected regions. It then triggers either load shedding to transfer the extra load to neighbor/non-neighbor servers or region aggregation to merge disconnected regions. While allowing load transfer between non-neighbor servers may improve the load balancing performance, spatial coherence within a partition may no longer be guaranteed. A region aggregation process needs to be executed occasionally to resolve this problem, resulting in a higher computation overhead.

In summary, existing local load balancing methods can efficiently resolve the load balancing problem. However, they tend to produce short-term load balancing solutions, as they only consider local load information when making the load balancing decisions. Hence, load balancing methods that not only produce good load balancing solutions, but also have low computation overheads are desirable for DVE applications.

2.3 Hybrid Load Balancing

The main limitation of global methods is that they are not very efficient, while the main limitation of the local methods is that their load balancing solutions are not very effective. To address the limitations of these two approaches, [Lau 2010] proposes a hybrid approach to address the DVE load balancing problem. It augments a local load balancing algorithm [Ng et al. 2002] with some global load information obtained less frequently. This global load information is used to guide the local load

balancing algorithm where to redistribute the loads so as to produce more long-term load balancing solutions. Results show that this approach helps reduce the average number of overloaded servers. However, this approach requires a dynamic server clustering step, which can be complex.

Unlike the hybrid approach, our heat diffusion based methods proposed here are very simple in concept and extremely efficient.

2.4 Heat Diffusion Algorithms

The heat diffusion process models heat diffusing from region of high temperature to region of low temperature over time. The diffusion follows the gradient of the temperature field and heat flows smoothly across regions of different temperature. The heat diffusion approach has been applied to guide load migration for solving dynamic load balancing problems in several domains, including multiprocessor computing, parallel finite element computing and parallel molecule dynamics simulation.

Heat diffusion was first presented in [Cybenko 1989] for load balancing on message passing multiprocessor networks. The method can achieve global load balance with very small computation and communication overheads as it involves only simple calculations and local load information. [Willebeek-LeMair and Reeves 1993] points out that the heat diffusion algorithm is superior to some global methods, like the gradient model algorithm [Lin and Keller 1987] and the hierarchical algorithm [Horton 1993], in terms of quality and speed. In addition, the heat diffusion algorithm has an optimal property that the amount of load needed to be migrated is minimized, which is comparable to the global optimization based methods [Hu and Blake 1998]. A limitation of the heat diffusion methods, as pointed out by [Boillat 1990], is that it may converge slowly for graphs with very large scale but very small connectivity. Some methods have been proposed to accelerate the convergence speed. [Watts and Taylor 1998] applies a second-order implicit scheme, instead of a first-order explicit scheme used in [Cybenko 1989], to shorten the convergence time. [Muthukrishnan et al. 1998] and [Hu and Blake 1999] use the Chebyshev polynomial to do so. Although these methods can reduce the convergence time, they have higher computation and communication overheads. Hence, for graphs with limited scale and reasonable connectivity, it is still better to employ the first-order explicit scheme, due to its simplicity. Although heat diffusion is proven to be very useful in several traditional areas, it has not yet been employed for DVE load balancing, which has a very high demand on interactivity.

3. OUR DYNAMIC LOAD BALANCING ALGORITHMS

We model the VE by dividing it regularly into a large number of square cells. Without loss of generality, we assume that the computation load of each cell is computed according to the number of users (or avatars) within it. In order to map the VE to multiple servers, it is divided into n partitions, referred to as $\{P_1, P_2, \dots, P_n\}$. If each partition is assigned to a separate server, there will be n servers referred to as $\{S_1, S_2, \dots, S_n\}$, with S_i serving partition P_i . For simplicity, we assume for the rest of this paper that references to a partition and the corresponding server are interchangeable if the context is clear. Each partition contains a number of cells and each cell may contain some users. The load of server S_i , denoted as l_i , is therefore the summation of the loads of all the cells inside partition P_i , counted by the number of users. Thus the load distribution of the n servers of the DVE system is $\{l_1, l_2, \dots, l_n\}$. Fig. 1(a) shows an example DVE.

Given a DVE system defined above, we may construct a server graph $G = (S, E)$, where S contains the set of nodes, i.e., servers, in the DVE system and E contains the set of edges with each edge linking two servers managing two adjacent partitions. The server graph is a weighted undirected graph. The weight associated with each node S_i is represented by its load l_i , and the weight associated with each edge $e_{i,j}$ is represented by a coefficient called *diffusion coefficient* denoted by $c_{i,j}$, which will be defined later. Fig. 1(b) shows the server graph of the example DVE in Fig. 1(a).



(a) A VE divided into 18×18 cells (thin borders) and 9 partitions (thick borders), with load values shown in brackets determined by the number of users (blue dots).

(b) The server graph of the VE shown in (a), with 9 nodes representing 9 partitions connected by edges and associated diffusion coefficients shown in brackets.

Fig. 1. An example DVE and the corresponding server graph.

The load balancing process is to migrate loads from the overloaded servers to the underloaded servers. The load on an overloaded server can be considered as heat, which is being diffused from this server outwards via its neighbor servers. This is analogous to the heat diffusion process, which is governed by a partial differential equation called *heat equation*:

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

where $u(x, y, t)$ is a temperature function that describes the variation of temperature across spatial location (x, y) at time t , and α is the thermal conductivity. The heat equation is used to determine the change in the temperature function over time as heat spreads throughout space. Solutions of the heat equation are characterized by a gradual smoothing of the initial temperature by the flow of heat from hotter to colder regions.

Heat diffusion was first used for load balancing in [Cybenko 1989], which models the dynamic load balancing problem in a distributed multiprocessor network using a discrete form of the heat equation based on the first-order explicit scheme. For each node of the server graph $G = (S, E)$, the execution of the heat diffusion method is divided into a sequence of iteration steps. At k^{th} iteration step, node S_i computes its load using the following equation, assuming that $l_i^{(0)}$ is the initial load of S_i :

$$l_i^{(k)} = l_i^{(k-1)} - \sum_j c_{i,j} (l_i^{(k-1)} - l_j^{(k-1)}) \quad (2)$$

where $l_i^{(k-1)}$ and $l_j^{(k-1)}$ are the loads of node S_i and its adjacent node S_j , respectively, at $(k-1)^{th}$ iteration step. $c_{i,j}$ is the diffusion coefficient that determines the portion of the load difference between S_i and S_j to be exchanged between them. Cybenko [Cybenko 1989] also discusses the necessary and sufficient conditions for the convergence of the diffusion method and comes up with the appropriate diffusion coefficients for those graphs with binary n -cube topology (e.g., rings) to guarantee the convergence of the iterative process. For generalized graphs, Boillat [Boillat 1990] proposes to compute the diffusion coefficients as follows:

$$c_{i,j} = \frac{1}{\max\{\deg(S_i), \deg(S_j)\} + 1} \quad (3)$$

where $\deg(S_i)$ is the degree of node S_i , i.e., the number of nodes adjacent to S_i , and $c_{i,j} = c_{j,i}$. The idea here is to increase the diffusion coefficient as the degrees of both adjacent nodes decrease.

In our DVE load balancing problem, the load needed to be transferred between any two neighbor servers, S_i and S_j , can be viewed as a flow of load along the edge $e_{i,j}$. We refer to this flow as a *balancing flow* and denote it as $\lambda_{i \rightarrow j}$. Note that if $\lambda_{i \rightarrow j}$ is a positive value, the flow direction is from S_i to S_j ; otherwise, it is from S_j to S_i . Then, our load balancing problem can be viewed as finding the balancing flow along each edge such that the load on each server becomes roughly the same:

$$\bar{l} = l_i - \sum_j \lambda_{i \rightarrow j} \quad (4)$$

where \bar{l} is the average load over all servers, i.e., $\bar{l} = (\sum_k l_k)/n$. $\sum_j \lambda_{i \rightarrow j}$ represents the summation of the balancing flows between S_i and each of its neighbor servers S_j .

According to the level of knowledge that the load balancing process needs in order to make load balancing decisions, we propose two methods based on the heat diffusion approach. The first one is the *local diffusion method*, in which every server that manages a partition computes the balancing flows based on the load information of its neighbor servers only. The second one is the *global diffusion method*, in which a central server computes the balancing flows for every server based on the load information of all the servers and delivers the computed balancing flows to the corresponding servers. Note that this central server can be any of the servers managing a partition or a dedicated server.

3.1 The Local Diffusion Method

Based on Eq. (2) and (3), server S_i computes a balancing flow for each of its neighbor servers S_j as:

$$\lambda_{i \rightarrow j} = c_{i,j}(l_i - l_j) \quad (5)$$

We note from Eq. (5) that server S_i only needs to have the load information of its neighbor servers in order to compute the balancing flows. After determining the balancing flows, S_i may start transferring its load to its neighbor servers. In practice, the transfer of load is application specific, due to the differences in the definition of load in different applications. In our DVE system, we subdivide the VE into cells and each cell is considered as a basic unit for load transfer among the servers. As the load in the VE is measured in terms of the number of users and 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 efficiently.

Our cell selection scheme can be divided into two processes: an initial process and a runtime process. During the initial process, for each server S_i , we construct a list of boundary cells with each of S_i 's neighbor servers, S_j . Each of these boundary cells has at least one side connected to S_j . We refer to this list as the *boundary cell list*. (Hence, if S_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 S_j . As there is at least one side and at most three sides of each cell connected to S_j , this priority value ranges between 1 to 3. Finally, we sort this list in descending order of the priority value.

During the runtime process, when server S_i has determined the balancing flow $\lambda_{i \rightarrow j}$, if its value is negative, it means that S_i will receive some load from S_j . Hence, it does not need to carry out the cell transfer. If its value is positive, we first round it up to the nearest integer number of users to be transferred, $\lceil \lambda_{i \rightarrow j} \rceil$. We then perform the following cell transfer 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.

Suppose that at the N^{th} iteration step ($N > 0$), we have:

$$L^{(N)} = \begin{pmatrix} l_1^{(N)} \\ l_2^{(N)} \\ \vdots \\ l_n^{(N)} \end{pmatrix} = \begin{pmatrix} \bar{l} \\ \bar{l} \\ \vdots \\ \bar{l} \end{pmatrix} \quad (7)$$

Then, the iterative computation process of Eq. (6) would terminate. In practice, we may not really need to achieve such an exact load balance. Since the load is measured in terms of the number of users, the computed load values must be rounded to the nearest integer numbers. Besides, an approximate balance will also allow the iterative computation process to terminate earlier. We will discuss this issue more in Section 3.3. Finally, we have the list of all intermediate load distribution vectors as follows:

$$\begin{cases} L^{(0)} = (l_1^{(0)}, l_2^{(0)}, \dots, l_n^{(0)})^T \\ L^{(1)} = (l_1^{(1)}, l_2^{(1)}, \dots, l_n^{(1)})^T \\ \vdots \\ L^{(N)} = (l_1^{(N)}, l_2^{(N)}, \dots, l_n^{(N)})^T \end{cases} \quad (8)$$

If we rearrange all the load values, we have the load vectors for each of the nodes (or local servers) over N iterations as follows:

$$\begin{cases} L_1 = (l_1^{(0)}, l_1^{(1)}, \dots, l_1^{(N)}) \\ L_2 = (l_2^{(0)}, l_2^{(1)}, \dots, l_2^{(N)}) \\ \vdots \\ L_n = (l_n^{(0)}, l_n^{(1)}, \dots, l_n^{(N)}) \end{cases} \quad (9)$$

where L_i contains the sequence of intermediate load values of server S_i . Each iteration step defined by Eq. (6) computes the amount of load that each node of the server graph, S_i , should transfer to (or receive from) each of its adjacent nodes. Thus, the total balancing flow across any edge $e_{i,j}$ is obtained by accumulating the intermediate loads transferred across $e_{i,j}$ over N iterations:

$$\lambda_{i \rightarrow j} = \sum_{k=0}^{N-1} c_{i,j} (l_i^{(k)} - l_j^{(k)}) \quad (10)$$

After determining the balancing flows for each local server, the central server will then send this information to the local servers. Upon receiving the balancing flow values, each local server will start the load transfer using the efficient cell selection scheme discussed in Section 3.1.

3.3 The Convergence of Balancing Flow Computation

As it is not necessary to achieve an exact load balance in practice, we introduce a convergence threshold here to control the level of load balance for our global diffusion method. When computing the balancing flows, the iterative process will stop if the deviation of the load of every local server from the mean load is smaller than the convergence threshold, θ_{conv} , as follows:

$$\frac{1}{\bar{l}} \left(\text{MAX}_{\text{for all } i} (l_i^{(k)} - \bar{l}) \right) < \theta_{conv} \quad (11)$$

where $\text{MAX}()$ finds out the maximum values from the input set. $l_i^{(k)}$ is the intermediate load of S_i at the k^{th} iteration step of the balancing flow computation process defined by Eq. (6). \bar{l} is the average load

of all local servers. Note that we model θ_{conv} as the percentage deviation from the average server load. In general, a higher convergence threshold value allows the iterative process to terminate earlier. As such, we expect it to affect the computation time as well as the amount of load needed to be transferred among the local servers. We will investigate how it affects our global diffusion method in Section 4.

The connectivity of the server graph also affects the convergence of the balancing flow computation. According to [Cybenko 1989], the condition needed to be hold for Eq. (6) to converge to the uniform distribution is that the server graph should be fully connected. In our DVE system, the VE is subdivided into cells which are then assigned to the partitions. As all the cells are connected, the partitions containing them are also connected, and so is the server graph. Regarding the convergence speed, [Boillat 1990] has proved that it is relatively slow in server graphs with low connectivity, and the worst situation occurs when the server graph is a ring. In our case, the server graphs are typically in a grid-like topology with moderate connectivity. Hence, the convergence problem is generally not a concern when adopting the heat diffusion approach in DVE load balancing.

3.4 The Load Balancing Interval

In a DVE system where all the local servers are placed in the same site, the communication delay among the servers may be neglected. However, in a large scale DVE system, users may be accessing the service from around the world, and the local servers may need to be physically distributed. In this situation, the communication delay among the local servers may no longer be neglected. The existence of this network delay causes two accuracy problems for the *global diffusion method*. First, as there is a time gap between the moment when a local server sends its load information to the central server and the moment that the local server receives the balancing flow values from the central server, the latest load of the local server may be very different from the one used by the central server to compute the balancing flows. This information aging problem is elaborated and addressed in [Deng and Lau 2012].

Second, if the communication delay among the local servers is comparable to the interval between two consecutive load balancing processes, the previous load balancing process may not have been completed when the central server begins a new load balancing process. This will cause significant load fluctuation. To address this problem, we need to make sure that the interval between any two consecutive load balancing processes, denoted as T_{intvl} , is longer than the time needed to complete a load balancing process:

$$\text{MAX}_{\text{for all } i} (\Delta t(S_i, S_C)) \times 2 + \text{MAX}_{\text{for all } i, j} (\Delta t(S_i, S_j)) + t_{comp} < T_{intvl} \quad (12)$$

where $\Delta t(S_i, S_C)$ is the single-trip delay between local server S_i and central server S_C . We multiply it by 2 to consider the round-trip delay here due to the time taken by S_i to send its load information to S_C and the time taken by S_C to send the balancing flow values to S_i . (To simplify our discussion, we simply assume that the delay from S_i to S_C is the same as that from S_C to S_i .) $\Delta t(S_i, S_j)$ is the single-trip delay between local server S_i and its neighbor local server S_j . This is the time taken by S_i to transfer its load to S_j . t_{comp} is the computation time taken by S_C to compute the balancing flows.

The second problem also affects the *local diffusion method*. Hence, we also need to make sure that the interval between any two consecutive load balancing processes of the *local diffusion method* is longer than the time needed to complete a load balancing process as follows:

$$\text{MAX}_{\text{for all } i, j} (\Delta t(S_i, S_j)) \times 2 + t_{comp} < T_{intvl} \quad (13)$$

We multiply $\Delta t(S_i, S_j)$ by 2 to consider the round-trip delay due to the time taken by S_j to send its load information to S_i and the time taken by S_i to transfer its load to S_j . (Here again, we assume that

the delay from S_i to S_j is the same as that from S_j to S_i .) t_{comp} is the computation time taken by S_i to compute the balancing flows.

This load balancing interval defines the shortest time interval that we may perform the load balancing process in the local or the global diffusion method without incurring the load fluctuation problem mentioned above.

4. EXPERIMENTAL RESULTS AND EVALUATIONS

To evaluate the performance of the proposed methods, we have implemented a simulation environment of a DVE in C++. The testing platform is a PC with an Intel Core i7 2.80GHz CPU and 6GB RAM. We have simulated a VE of $1,000m \times 1,000m$ in size. It is subdivided into a set of 100×100 cells. Hence, the size of each cell is $10m \times 10m$. The cells are then divided into 100 partitions and managed by 100 servers. The load of a partition is quantified by the number of users within it. We assume that each server can serve a maximum of 50 users without being overloaded. (Note that this is a tunable parameter depending on the computation performance of the server.) Fig. 2 shows a screenshot of a part of the simulated VE, with each dot representing a user.

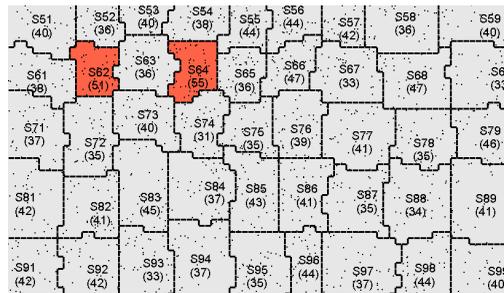


Fig. 2. A screenshot of a part of the virtual environment. The two numbers within each partition indicate the server ID (outside braces) and the current load value (inside braces), i.e., the number of users. The two partitions in red color represent overloaded regions, as their load values exceed 50.

To simulate the movement of users in the VE, we may adopt the *random waypoint mobility model* [Johnson and Maltz 1996], which is widely used in mobile networking research for testing routing algorithms. The key feature of this mobility model is that the next destination, called a waypoint, to which a user will head to as he reaches the current one is a randomly chosen location within the VE. However, this model may not be accurate enough to model avatar mobility in DVEs, as pointed out by [Liang et al. 2009] based on their measurement study in Second Life and [Pittman and GauthierDickey 2010] based on their measurement study in World of Warcraft. [Pittman and GauthierDickey 2010] suggests that a non-uniform distribution like the log-normal distribution (similar to the power law distribution) of waypoint choices, rather than the uniform distribution used in the random waypoint mobility model, is more accurate for avatar mobility modeling. Based on this suggestion, we introduce some hotspots to the VE as potential waypoints. In our simulation, we follow the 80-20 rule of the power law that we set the number of randomly distributed hotspots to 20% of the total number of partitions and we set the probability for a user to choose anyone of the hotspots as his waypoint to 80% and a randomly generated waypoint to 20%. We have created 4,000 users to move around in the VE, whose initial locations are randomly chosen. Each user repeatedly selects a waypoint and then moves to it with a speed randomly chosen from a speed range. We use two speed ranges in our experiments: a slow speed range of $[0, 3m/s]$ to model human walking speeds, referred to as the *slow scenario*, and a

high speed range of $[0, 10m/s]$ to model human running speeds, referred to as the *rapid scenario*. The slow scenario is to test the load balancing method during normal situation, while the rapid scenario is to test the load balancing method under a more extreme situation. In particular, the rapid scenario may be used to simulate a demanding situation of a massively multiplayer online game (e.g., World of Warcraft), while the slow scenario may be used to simulate a less demanding situation of a social network virtual world (e.g., Second Life), according to the report in [Machado et al. 2010].

To study how non-uniform the users are distributed in our simulation, we have captured the number of users (population) in each partition. Here, we spatially divide the entire VE evenly into 100 partitions. Fig. 3 shows the cumulative distribution function (CDF) of the partition population for each scenario. In the two diagrams, the blue dash-dot curve represents our user distribution while the red solid curve represents the fitted Weibull distribution. According to [Pittman and GauthierDickey 2010], the Weibull distribution has been found to be an appropriate model to describe the partition population distribution. From Fig. 3(a) and 3(b), we can see that our user distribution is well fitted to the Weibull distribution.

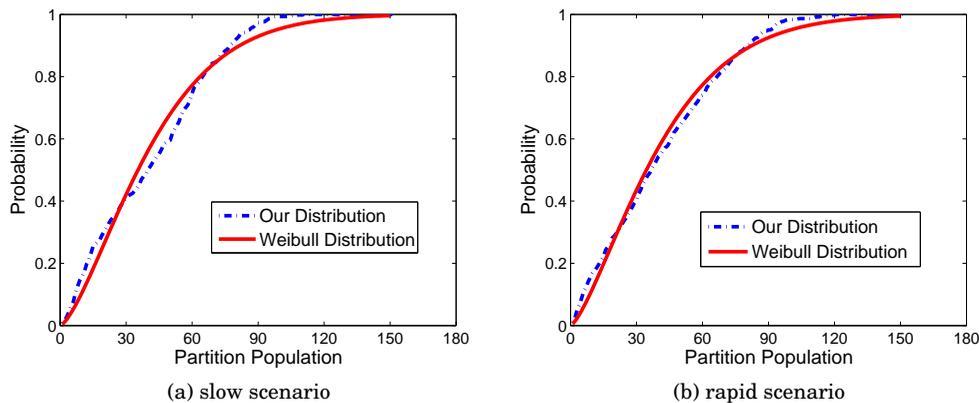


Fig. 3. Cumulative distribution of partition population.

The system frame rate is set to 10. Hence, the system updates the location of each user in the VE every $0.1s$. The total duration of the experiments is 1,000 seconds. Typically we run the simulation for a while so that the DVE system is stabilized before we begin to capture the performance information. We use the following metrics to evaluate the performance of the load balancing methods:

- Number of overloaded servers (N_{OS}): This counts the number of servers getting overloaded within a certain period of time. 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 method in avoiding server overloading and should be as low as possible.
- Number of migrated users (N_{MU}): This counts the number of users being migrated per second. Migrating a large number of users consumes network bandwidth and may affect the interactivity experienced by the users being migrated. This metric is to evaluate the transmission overhead induced by the load balancing method and should be as low as possible.

4.1 Experiment #1: Comparison on Load Balancing Performance

In this experiment, we compare the performance of the two proposed load balancing methods with three existing methods: the adaptive partitioning method [Ng et al. 2002], which is a purely local method, the hybrid load balancing method [Lau 2010], which is based on [Ng et al. 2002] but with added global information to guide load redistribution, and the method proposed in [Chen et al. 2005], which is similar to [Ng et al. 2002] but allows load migration from an overloaded server to non-neighbor underloaded servers. To simplify the discussion, we refer to our local diffusion method as *Local*, our global diffusion method as *Global*, the adaptive partitioning method [Ng et al. 2002] as *Adaptive*, the hybrid method [Lau 2010] as *Hybrid*, and the method in [Chen et al. 2005] that allows remote load migration as *Remote*. We ignore the network delay among the servers in this experiment by assuming that they are located within the same LAN, so that we may focus on comparing the load balancing performance of these methods. We experiment two load balancing intervals, $0.1s$ (or 1 frame) and $1s$ (or 10 frames). In *Global*, the convergence threshold θ_{conv} as discussed in Section 3.3 is set to 0.2 and 0.1 when the load balancing interval is set to $0.1s$ and $1s$, respectively. These are the optimal values that we obtain from Experiment #4 to be discussed in Section 4.4. In this experiment, both the slow and the rapid scenarios are tested.

Fig. 4 compares the N_{OS} and N_{MU} performances of the five methods when the load balancing interval is set to $0.1s$ (i.e., when the load balancing process is executed once per frame). Fig. 4(a) shows N_{OS} performance of each method. We can see that *Adaptive* has the highest average N_{OS} value than the other methods. This shows that without any global information, *Adaptive* is not able to redistribute the load well and hence causes frequent server overloading. With the guidance of global information in load redistribution, *Hybrid* performs better than *Adaptive*. *Remote* allows load migration among non-neighbor servers, making it easier to resolve server overloading. We can see that *Remote* has lower average N_{OS} value than those of *Adaptive* and *Hybrid*. On the other hand, *Local* and *Global* have no server overloading observed during the whole period, i.e., their average N_{OS} values are zero. This indicates that the heat diffusion approach is effective in DVE load balancing. Although *Local* is a local method, it can diffuse the load among the servers over time effectively. We have also measured the standard deviation of the N_{OS} values for each method as shown in Fig. 4(b) for reference. Fig. 4(c) shows the corresponding N_{MU} performance of each method. We can see that *Adaptive* also has a higher N_{MU} value than the other four methods. This is because if the servers get overloaded frequently, more users will need to be migrated among the servers. The other four methods have similar N_{MU} performance. We may also see that *Local* has a larger N_{MU} value than *Global*. This is because *Local* only performs one iteration during each load balancing process, while *Global* performs many iteration (until converged), which helps reduce load discrepancy among servers in a shorter time. Again, we show the standard deviation of the N_{MU} values of each method in Fig. 4(d). From Fig. 4(a) and 4(c), we observe that as the users move faster (in the rapid scenario), the average N_{OS} and N_{MU} values for all methods become higher, compared to those in the slow scenario. This is because as the users move faster, it generally takes a shorter time for the users to move across partition boundaries, and hence for the servers to get overloaded. Overall, this part of the experiment shows that the heat diffusion approach for DVE load balancing is more effective, with *Global* outperforming the other methods.

Fig. 5 shows similar experimental results as Fig. 4, except that the load balancing interval is now set to $1s$ (i.e., when the load balancing process is executed once every 10 frames). This is to study the N_{OS} and N_{MU} performances of the five methods as we execute the load balancing process less frequently. Fig. 5(a) shows that all five methods now have higher average N_{OS} values compared to Fig. 4(a). This is mainly because the loads of the servers become more imbalanced as we execute the load balancing process less frequently. In particular, *Adaptive*, *Hybrid* and *Remote* now have much higher average

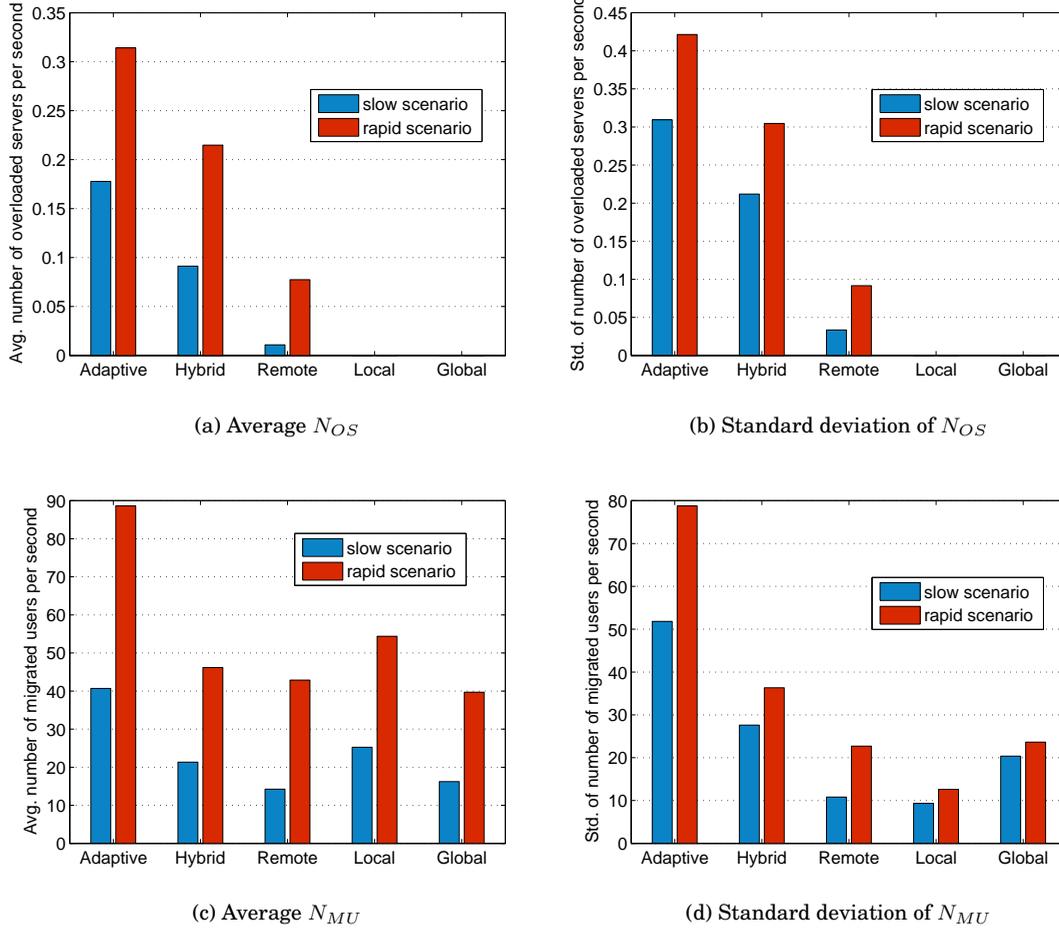


Fig. 4. Comparison of the N_{OS} and N_{MU} performances, when $T_{intvl} = 0.1s$.

N_{OS} values in the rapid scenarios, while *Local* and *Global* begin to see overloaded servers in the rapid scenario. Fig. 5(b) shows the standard deviation of the N_{OS} values for each method for reference. Fig. 5(c) shows that the average N_{MU} are all lower than those in Fig. 4(c). This is due to the reduced number of load balancing processes from 10 per second to 1 per second. We can also see that all five methods now have similar N_{MU} values. Fig. 5(d) shows the standard deviation of the N_{MU} values for each method.

Overall, this experiment shows that the heat diffusion approach for DVE load balancing is more effective than *Adaptive*, *Hybrid*, and *Remote*. They can significantly reduce the number of overloaded servers. In particular, *Global* outperforms all the other methods.

4.2 Experiment #2: Comparison on Computation Time

In this experiment, we compare the computation times of the five methods. Table I shows the average computation time of each load balancing process for each method when the load balancing interval is set to $0.1s$ and $1s$. We can see that the computation times of our proposed methods *Local* and *Global*

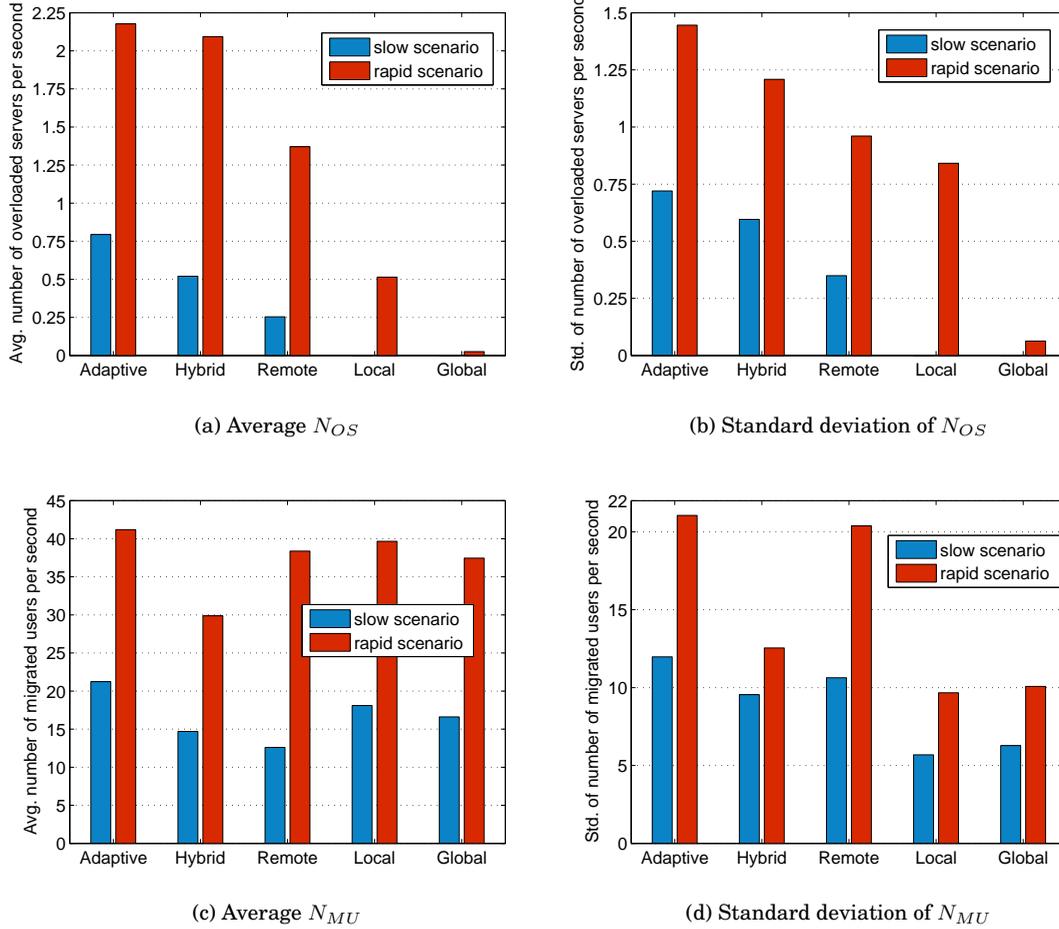


Fig. 5. Comparison of the N_{OS} and N_{MU} performances, when $T_{interval} = 1s$.

are comparable to those of *Adaptive* and *Hybrid*. *Remote* has a relatively higher (roughly ten times higher) computation time than the others. This is mainly because *Remote* has an extra partition aggregation process to correct the isolated locality disruption caused by remote cell migration between non-neighbor partitions. We may also see that there is a small increase in computation times as we increase the moving speed from the slow scenario to the rapid scenario or the load balancing interval from $0.1s$ to $1s$. The reason may be that either increasing the moving speed or the load balancing interval causes a higher degree of load imbalance. As a result, it takes longer for the methods to achieve load balance. Overall, our proposed methods are efficient compared to existing methods and can be executed in every frame without affecting the interactivity of the DVE system.

4.3 Experiment #3: Performance Against Network Latency

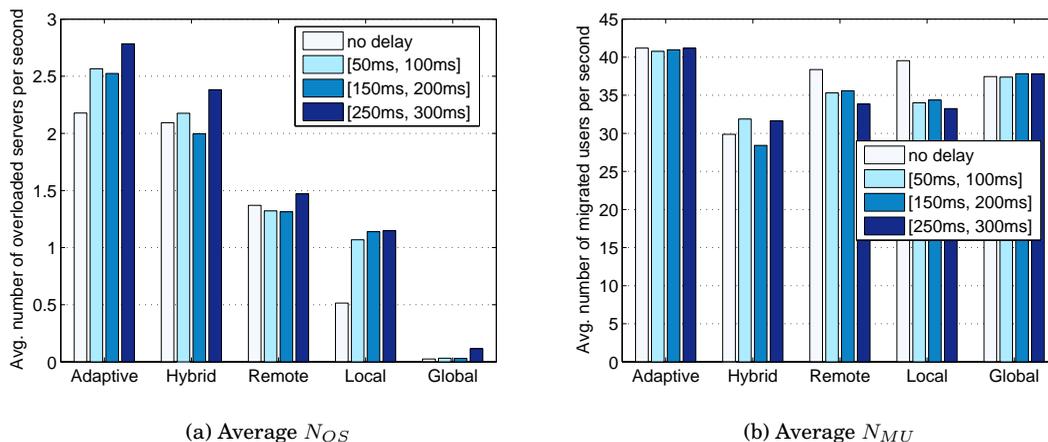
In order to apply the proposed load balancing methods in a real world DVE system, where some of the servers may be distributed across the globe, we need to consider the impact of network delay existed among the servers. This experiment studies the performance of our two methods in the presence of

Table I. The average computation time (in *ms*) of each load balancing process.

	slow scenario					rapid scenario				
	<i>Adaptive</i>	<i>Hybrid</i>	<i>Remote</i>	<i>Local</i>	<i>Global</i>	<i>Adaptive</i>	<i>Hybrid</i>	<i>Remote</i>	<i>Local</i>	<i>Global</i>
$T_{intvl} = 0.1s$	1.1	1.2	13.4	1.0	1.6	1.1	1.2	14.1	1.0	1.6
$T_{intvl} = 1s$	1.2	1.4	14.8	1.0	1.6	1.5	2.1	15.0	1.0	1.8

network delay. We consider four ranges of network delay: no delay, $[50ms, 100ms]$, $[150ms, 200ms]$, and $[250ms, 300ms]$. The first one is an ideal situation where there is no delay among the servers. The last one, i.e., $[250ms, 300ms]$, represents a high latency environment and models the situation where the servers are connected through the Internet. To set T_{intvl} , we may consider *Global*, which is most affected by the network delay. The maximum time needed to run the whole load balancing process is around $900ms$ when considering the above delay settings according to Eq. (12). Hence, we may safely set the load balancing interval to $1s$. For the convergence threshold, we use the same setting as in Experiment #1, i.e., $\theta_{conv} = 0.1$ for *Global*.

Fig. 6 compares *Local* and *Global* on the average N_{OS} and N_{MU} values under different ranges of network delay. (We have also included the corresponding values from *Adaptive*, *Hybrid* and *Remote* here for reference.) Fig. 6(a) shows that as the network latency increases, the average N_{OS} values of all methods also increase in general. This is because the load information that the servers receive for computing the balancing flows ages as the network latency increases. Hence, the load balancing solutions produced become less accurate, causing more servers to get overloaded. Fig. 6(b) shows that as the network latency increases, the average N_{MU} value of *Local* appears to be decreasing while that of *Global* does not change much. In both situations, the range of variation is within around 15%, which is too small to give a conclusion. Hence, we have conducted a similar experiment with $T_{intvl} = 3s$ and more ranges of network delay. We have found that when the load balancing interval is fixed, the average N_{MU} values of both methods only fluctuate within a small range as we increase the network latency. Similar results can also be observed from the other three methods. Hence, we may conclude that the average N_{MU} values are not affected by the network latency. This may be explained as follows. When the load balancing solutions are not accurate due to load information aging, some servers may actually transfer more load than necessary while other may transfer less load than required. Hence, the average number of migrated users remains more or less the same.

Fig. 6. N_{OS} and N_{MU} performances under different ranges of network delay.

4.4 Experiment #4: Performance Against Convergence Threshold

In this experiment, we study the effect of the convergence threshold, θ_{conv} , on our global diffusion method in terms of N_{OS} and N_{MU} . Fig. 7(a) shows that as we increase the convergence threshold, the N_{OS} value also increases. This is because increasing the convergence threshold allows higher load deviations among the servers, making it easier for the servers to get overloaded. We can also see that if we increase the load balancing interval from $0.1s$ to $1s$, the N_{OS} value also increases, which agrees with the results shown in Experiment #1. Fig. 7(b) shows that as we increase the convergence threshold, the N_{MU} value decreases. This is because increasing the convergence threshold allows higher load deviations among the servers and hence requires less load to be transferred in order to achieve a load balancing state. In addition, if we increase the load balancing interval from $0.1s$ to $1s$, the N_{OS} value also increases. This is due to the increase in load deviation among the server as a result of the increased load balancing interval.

Overall, we have found that the total number of migrated users is very small, even for small convergence threshold values. In practice, if we duplicate all user information, including the geometry representing the avatars, in each local server, we only need to transfer an average of 48 user IDs and locations among the local servers when $\theta_{conv} = 0.05$ and $T_{intvl} = 0.1s$. This is only a small amount of data for transmission. Hence, to determine an appropriate convergence threshold value, we only need to consider the number of overloaded servers in Fig. 7(a). When $T_{intvl} = 0.1s$, we find that there is a critical point when the convergence threshold value is at around 0.2. From 0 to 0.2, the slope of the curve is very gentle. After 0.2, the slope increases significantly. Hence, the optimal convergence threshold value would be 0.2 when the load balancing interval is set at $0.1s$. Likewise, the optimal convergence threshold value would be 0.1 when the load balancing interval is set at $1s$.

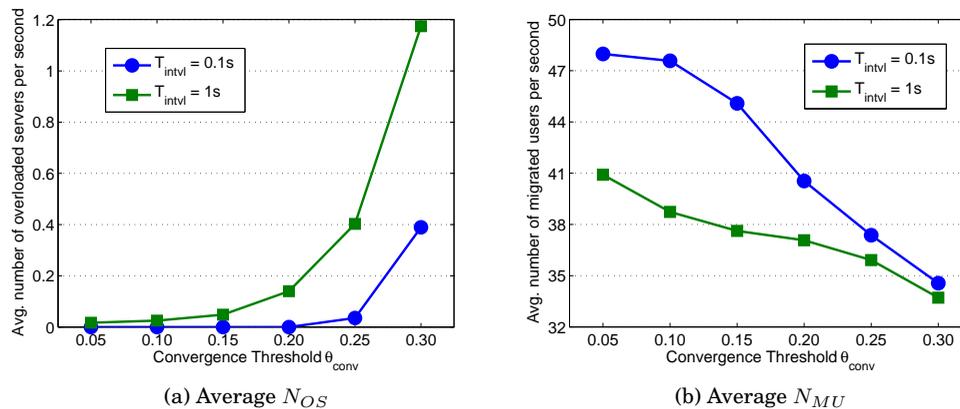


Fig. 7. N_{OS} and N_{MU} performances of our global diffusion method for various θ_{conv} settings.

4.5 Evaluations

We have compared our two heat diffusion based methods with three existing DVE load balancing methods. Results show that our methods are more effective than these methods in reducing the numbers of overloaded servers. In general, they also produce low numbers of migrated users. Even though our methods are more effective, their computational costs are comparable to the other methods. In terms of implementation efforts, these existing methods that we compare with are all based on heuristics.

There are a lot of exceptional cases needed to be handled. On the other hand, the heat diffusion based methods have very simple concept and there is no need to handle any exceptional cases.

Comparing the two proposed methods, the main difference between the two methods is that in each load balancing process, the local diffusion method computes a single iteration of load balancing flows and then executes the load balancing flows, while the global diffusion method computes many iterations of load balancing flows until the results converge before executing the load balancing flows. From our experimental results, the global method has a lower number of overloaded servers than the local method, while the two methods have similar numbers of migrated users. The computation overheads of both methods are very small. Overall, the global method has a clear performance advantage over the local method.

However, the global method also has its limitations compared with the local method. First, it requires a central server/process to collect load information from all servers, compute the load balancing flows and then distribute the load balancing flow values to the local servers. This central server/process can potentially become a single point of failure. One simple solution to address this limitation is that whenever the DVE system detects a failure in the central server/process, each local server may initiate the local diffusion process to perform local load balancing. Due to the similarity of the two methods, such a dynamic change in the load balancing methods is expected to be straightforward. Second, the global method suffers a higher overall delay (i.e., a single-trip delay to collect load information from all servers, a single-trip delay to distribute the load balancing flows to all servers and a single-trip delay to perform the load balancing with neighbor servers) than the local method (i.e., a single-trip delay to collect the load information from the neighbor servers and a single-trip delay to perform the load balancing with the neighbor servers), as discussed in Section 3.4. This overall delay affects the setting of the load balancing interval. Our experimental results show that a higher load balancing interval leads to a small increase in the number of overloaded servers.

Server overloading occurs as a result of users moving around causing uneven load distribution among the servers. A higher user mobility leads to a higher uneven load distribution and therefore a higher chance of server overloading. There are different types of DVE applications. Some may have a higher user mobility, causing a higher computation overhead. Massively multiplayer online games are generally considered as the most demanding applications due to their high user mobility. As we have demonstrated in this paper, our proposed methods perform well in the extreme situation where users move rapidly. We believe that these methods can also be applied to other DVE applications such as social network virtual worlds, which in general have relatively lower user mobility.

5. CONCLUSION

In this paper, we have proposed a new dynamic load balancing approach for DVEs based on the heat diffusion model. We have investigated two dynamic load balancing methods based on the approach, a local load balancing method and a global load balancing method. In the local method, each local server performs load balancing with its own neighboring servers based on their load information. In the global method, a central server/process determines how each local server should perform the load balancing using the load information of all servers and then instructs each local server to carry out the suggested load balancing with its neighboring servers. We have analyzed two practical factors of the proposed methods, the convergence threshold and the load balancing interval.

The main advantage of our heat diffusion based load balancing approach is that it is extremely simple. As a result, it is also very efficient as demonstrated by our experimental results. In addition, our results show that the two proposed load balancing methods perform better than some existing methods, in terms of the number of overloaded servers and the number of migrated users. Overall, the global method performs better than the local method.

Acknowledgments

We would like to thank all the anonymous reviewers for their helpful and constructive comments. The work presented in this paper was partially supported by two GRF grants from the Research Grants Council of Hong Kong (RGC Reference Numbers: CityU 116010 and CityU 115112).

REFERENCES

- BOILLAT, J. 1990. Load balancing and poisson equation in a graph. *Concurrency: Practice and Experience* 2, 4, 289–313.
- 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 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.
- DENG, Y. AND LAU, R. 2010. Heat diffusion based dynamic load balancing for distributed virtual environments. In *Proc. ACM Symp. on VRST*. 203–210.
- DENG, Y. AND LAU, R. 2012. On delay adjustment for dynamic load balancing in distributed virtual environments. *IEEE Trans. on Visualization and Computer Graphics (special issue of Proc. IEEE VR 2012)* 18, 4, 529–537.
- HORTON, G. 1993. A multi-level diffusion method for dynamic load balancing. *Parallel Computing* 19, 2, 209–218.
- HU, Y. AND BLAKE, R. 1998. The optimal property of polynomial based diffusion-like algorithms in dynamic load balancing. In *K.D. Papailiou et al. (Ed.), Computational Dynamics'98*, Wiley. 177–183.
- HU, Y. AND BLAKE, R. 1999. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing* 25, 4, 417–444.
- HU, Y., BLAKE, R., AND EMERSON, D. 1998. An optimal migration algorithm for dynamic load balancing. *Concurrency: Practice and Experience* 10, 6, 467–483.
- JOHNSON, D. AND MALTZ, D. 1996. Dynamic source routing in ad hoc wireless networks. *Mobile Computing* 353, 153–181.
- LAU, R. 2010. Hybrid load balancing for online games. In *Proc. ACM Multimedia*. 1231–1234.
- 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 Symp. on VRST*. 160–168.
- LIANG, H., DE SILVA, R., OOI, W., AND MOTANI, M. 2009. Avatar mobility in user-created networked virtual worlds: measurements, analysis, and implications. *Multimedia Tools and Applications* 45, 1, 163–190.
- LIN, F. AND KELLER, R. 1987. The gradient model load balancing method. *IEEE Trans. on Software Engineering* 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.
- MACHADO, F., SANTOS, M., ALMEIDA, V., AND GUEDES, D. 2010. Characterizing mobility and contact networks in virtual worlds. In *Proc. International Conference on Facets of Virtual Environments*. 44–59.
- 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 Symp. on VRST*. 163–170.
- OU, C. AND RANKA, S. 1997. Parallel incremental graph partitioning. *IEEE Trans. on Parallel and Distributed Systems* 8, 8, 884–896.
- PITTMAN, D. AND GAUTHIERDICKEY, C. 2010. Characterizing virtual populations in massively multiplayer online role-playing games. In *Proc. Int'l Conf. on Advances in Multimedia Modeling*. 87–97.
- PRASETYA, K. AND WU, Z. 2008. Performance analysis of game world partitioning methods for multiplayer mobile gaming. In *Proc. ACM Workshop on Network and System Support for Games*. 72–77.
- STEED, A. AND ABOU-HAIDAR, R. 2003. Partitioning crowded virtual environments. In *Proc. ACM Symp. on VRST*. 7–14.
- TA, D., ZHOU, S., CAI, W., TANG, X., AND AYANI, R. 2009. Efficient zone mapping algorithms for distributed virtual environments. In *Proc. ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*. 137–144.
- TA, D., ZHOU, S., CAI, W., TANG, X., AND AYANI, R. 2011. Multi-objective zone mapping in large-scale distributed virtual environments. *Journal of Network and Computer Applications* 34, 2, 551–561.
- VAN DEN BOSSCHE, B., DE VLEESCHAUWER, B., VERDICKT, T., DE TURCK, F., DHOEDT, B., AND DEMEESTER, P. 2009. Autonomic microcell assignment in massively distributed online virtual environments. *Journal of Network and Computer Applications* 32, 6, 1242–1256.
- WATTS, J. AND TAYLOR, S. 1998. A practical approach to dynamic load balancing. *IEEE Trans. on Parallel and Distributed Systems* 9, 3, 235–248.

0:20 • Y. Deng and R. Lau

WILLEBEEK-LEMAIR, M. AND REEVES, A. 1993. Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans. on Parallel and Distributed Systems* 4, 9, 979–993.

Received July 2012; revised May 2013; accepted June 2013