ABSTRACT—A distributed virtual environment (DVE) allows multiple geographically distributed objects to interact concurrently in a shared virtual space. Most DVE applications use a non-replicated server architecture, which dynamically partitions a virtual space. An important issue in this system is effective scalability as the number of users increases. However, it is hard to provide suitable load balancing because of the unpredictable movements of users and hot-spot locations. Therefore, we propose a mechanism for sharing roles and separating service regions. The proposed mechanism reduces unnecessary partitions of short duration and supports efficient load balancing.

Keywords—Distributed virtual environments, load balancing, partitioning algorithm, scalability issue.

I. Introduction

An important requirement of a distributed virtual environment (DVE) system is to provide its users with realistic interaction. This necessitates that the system has good load balancing. As the number of users increases, the DVE system faces some challenging problems, such as overhead, bottlenecks, and so on. There are several research works which partition the global game space into sub-spaces to solve such problems. In [1], microcells were proposed, which can be dynamically assigned to a set of servers to redistribute the load on servers. However, frequent re-mapping and region migration in the method leads to high overhead for servers. Also, in [2], a locality aware load balancing method was proposed which reduces cross-server communication. Though it especially considers awareness of spatial locality in a virtual space, the method also leads to frequent region migrations. Therefore, we propose a new mechanism for sharing roles and separating service regions (SRSS) which reduces unnecessary partitions of short duration.

II. SRSS Mechanism for Load Balancing of a DVE Server

To redistribute the load over several servers, a metric is needed to determine the load on servers. We propose the queue length for request packets (LRP) as the metric. The LRP is the number of tasks in the CPU queue residing in a process. Table 1 shows the three server status levels: light load, normal load, and heavy load. The upper threshold and lower threshold are denoted by $T_{up}$ and $T_{low}$, respectively.

We propose new partitioning condition parameters for consideration of historical data of players. We use a set of cells (such as 4×4 cells in Fig. 1) as the unit of historical data. The set of cells is called an upper grid. This is the minimum size for partitioning virtual space. Figure 1 shows the proposed parameters for partitioning virtual space. The vertical partition $S_1$, which assigns similar areas to two servers, is better than the horizontal partition $S_2$. Also, partition $S_4$ is better than partition $S_3$.
S3 in terms of evenly partitioning objects. Most objects in partition S4 were concentrated near the line of partition. Such objects have a high probability of migration from the current server to the other neighboring server. Therefore, the partition S4 leads to a high cost of server migration (SM). It was better to divide the virtual space at the point which yields the smallest number of objects near the line of partition. Also, the longer the border line, the higher the cost of SM. So, partition S6 is better than partition S5. Further, we consider the main paths of objects. Partition S7 is better than partition S8, for which the many main paths of the objects are cut by the line of partition. The main paths of objects are generated by summarizing the trajectories of objects based on the upper grid. However, because the partition procedure using historical data in real-time involves a high overhead, we partition the virtual space according to the proposed parameters when DVE systems are initiated. We evaluated our method except the main path of objects in [3], which provides an optimized equation for partitioning the virtual space.

To solve problems of hot-spot location that result from unpredictable movements of a massive number of users, we propose an SRSS mechanism with two levels: level-SR (sharing roles) and level-SS (separating service regions). The SRSS mechanism initially applies level-SR to heavily loaded servers. If it cannot solve the overload problems, level-SS is then applied. The first level, level-SR moves the parts of modules from heavily loaded servers to lightly loaded servers, without repartitioning virtual space. DVE servers have three main modules Packet_Receiver, which receives requested queries of users, Packet_Processor, which processes requested queries of users, and Packet_Sender, which sends results to users. The Packet_Receiver and Packet_Sender modules (Packet IO module) use a substantial proportion of the total run-time of the DVE server. Also, the Packet IO module does not need synchronization among servers. Therefore, we use it for sharing functions of level-SR. Figure 2(a) shows the procedures of level-SR. The following procedure is repeated until server A has the status of a normal load.

1. The heavily loaded server A initially selects players to be applied by level-SR (SRPlayers) based on the probability of server migration. SRPlayers disconnect from heavily loaded server A.

2. SRPlayers connect to lightly loaded server B. Server B receives the request packets from individual SRPlayers.

3. Server B combines the received request packets into one packet and sends it to server A. Server A sends a set of result packets of SRPlayers to server B.

4. Server B delivers the result packets to individual SRPlayers.

Figure 2(b) shows the assignment sequence of upper grids for selecting players to be applied by level-SR. Sequences are made according to the probability of server migration and locations of the server’s virtual space, namely, the corners, the center, and the sides. However, when level-SR is unable to solve the hot-spot location problems, we finally apply level-SS to the server. Level-SS involves placing regions of the overloaded server in a bin for assignment to the server with the lightest load with the sequence of upper grids used at level-SR as shown in Fig. 2(c). The SRSS method is executed in sub-linear time $O(gn)$, in terms of the number of upper grids(g) involved in a server, and the number of server participants (n).

III. Performance Evaluation

We evaluated our mechanism in single-server mode and multiple-server mode under various conditions.
In the experimental environments, virtual players (VP) are able to randomly move, seek monsters (within 5×5 cells), and fight monsters, which exhibit simple AI actions (move, avoid, attack, path finding). VPs have a screen view (8×8 cells) and can have a special destination for creation of a flocking. VPs periodically send location data and action data (the packet size is 1,024 bytes). VPs receive the status data of neighbors (8×8 cells) from the server. The size of the virtual space of a server is 1,024 bytes). VPs periodically receive the status data of their neighbors (8×8 cells) from the server. The size of the virtual space of a server is 1,024 bytes). VPs periodically receive the status data of their neighbors (8×8 cells) from the server. The size of the virtual space of a server is 1,024 bytes).

IV. Conclusion

This paper presented a load balancing mechanism which reduces unnecessary partitions, using new partitioning parameters, such as the number of objects near the line of partition, length of the line of partition, and the main path of an object. Simulation results demonstrate that our method is more efficient than the existing methods. In the near future, our mechanism will be applied to real applications for analysis of its performance.

References