2010 14th IEEE/ACM Symposium on Distributed Simulation and Real-Time Applications

Scale Virtual Worlds Through Dynamic Load Balancing

Huaiyu Liu and Mic Bowman
Intel Labs, Intel Corporation
Hillsboro, OR
{huaiyu.liu, mic.bowman}@intel.com

***Abstract*—Dynamic load balancing holds the potential to scale virtual worlds flexibly by dynamic allocation of hardware to match load. In this paper, we study the benefits and overheads of space based load partitioning, in particular, distributed binary space partitioning (BSP). Our evaluation is based on OpenSimulator, a virtual world system compatible with Second Life® viewers. Our work reveals that although simple and effective, distributed BSP has several limitations and suffers from high overhead. We then analyze the fundamental reasons of these limitations. To overcome the limitations, we argue that it is necessary to break away from the simulator-centric architecture used in today's virtual worlds, and present potential new directions.**

Keywords - virtual worlds; distrbuted simulation, load balancing; binary space partitioning

I. INTRODUCTION

Virtual worlds use simulation to create a fully-immersive 3D space in which users collaborate and interact in real time. More and more people and companies are using virtual worlds for training, collaboration, and sharing experiences in a way they could never before. It is estimated that there are 137 million virtual world users today, with up to 1 billion projected users in the year 2017 [1].

Scalability is still a great challenge in virtual worlds. To meet the increasing demand of rich user experiences, high level of realism, and new usages such as experiencing professional sports games in virtual stadiums, virtual worlds must scale beyond their current capability. For instance, existing virtual worlds such as Second Life® [2] and World of Warcraft™ [3] are limited to below 100 participants interacting with each other to provide satisfying user experiences [11]. Compared to traditional simulations, there are three unique requirements that pose great challenges in scaling virtual worlds: large scale, real-time and perpetual simulations with distributed interactions; visualization for many users, each with a unique perspective; and multiple heterogeneous simulation engines (which we refer to as "*actors*", for instance, physics and script engines).

Load balancing is one essential solution to sustain long-running distributed simulations and to scale flexibly by dynamic allocation of hardware. Some systems such as World of Warcraft use sharding to balance load. A shard is a copy of one part of a virtual world and different shards reside on different servers. Such a system scales at the price of user interaction: users on different shards cannot interact with each other. Other systems such as Second Life statically decompose the space into fixed size regions, and limit the scene complexity within each region and the number of users connecting to each region. In general, static load balancing suffers from over-provisioning or under-provisioning, since runtime workload is dynamic and hard to predict. Studies show that dynamic load balancing outperforms static solutions in both LAN and WAN settings [20], especially in hotspot or player flocking scenarios.

Virtual worlds have both computation and communication scalability bottlenecks [9][15]. We are working on solutions to address the communication bottleneck by offloading client management to separate hardware (client managers) [9], and will present the work in a companion paper. In this paper, we focus on addressing the computation bottleneck, and in particular, on managing and balancing the load of actor operations and management of objects contained in a 3D virtual space (load balancing client connections, however, will be handled separately by client managers [9]). There are three main contributions in this paper.

First, we evaluate the benefits and overheads of applying distributed binary space partitioning (BSP) to dynamically balance workload in virtual worlds. Our goal is to identify fundamental limitations and performance tradeoffs of dynamic partitioning with real scenarios running on live servers. Our evaluations are based on a prototype implemented on top of OpenSimulator [4], an open source platform for virtual world servers that is compatible with Second Life viewers. We have also developed synthetic workloads, which emulate workload dynamics, and applied real workloads from ScienceSim [7] with large volume of objects. Our analysis shows that distributed BSP is effective in balancing workload dynamically and alleviate hotspot problems. However, it has a few limitations: (1) high overhead of workload migration and (2) high overhead of inter-region communication, especialy with workloads that have disproportional density.

Second, we analyze the overhead of workload migration, especially object migration, during dynamic load balancing. Object migration has been largely overlooked in most previous studies. (Systems such as Second Life and World of Warcraft avoid workload migration by static and strict partitioning. Academic studies, on the other hand, tend to focus on migration of client connections and ignore object migrations.) We show that if not carefully designed and implemented, workload migration could be very expensive and result in poor system performance and user experiences.

1550-6525/10 $26.00 © 2010 IEEE
DOI 10.1109/DS-RT.2010.14

We present a method, "migration with database support", for optimizing the process. Our results show potentially orders of magnitude reduction on the object unavailable time during object migrations.

Third and most importantly, we discuss the limitations of distributed BSP and analyze the fundamental reasons for these limitations. A major reason is the simulator-centric architecture used in many virtual worlds. It views virtual world operations as a set of homogenous simulators, each aggregating the data structure and all the actors operating on the data structure. We show that this architecture tends to produce high overhead of workload migration, inefficient partitions, and has limited ability to apply "appropriately configured" hardware to the heterogeneous computation and communication tasks. We argue that a different architecture is necessary to address the limitations and meet the requirements of load balancing in virtual worlds. We also present potential directions for future research.

Several dynamic load balancing algorithms have been proposed in the context of virtual worlds [13][15][18][20][21][22][23][24]. These studies mostly focused on balancing the server load in terms of the number of clients per server or organizing clients into multicast groups to reduce network traffic. There are several limitations. First, they overlook the fact that operations in a virtual world not only include managing players and controlling traffic to client machines, but also include operations of other actors such as physics engine and script engine. These actors have different performance constraints and hence may prefer different load balancing policies. Second, they tend to overlook the cost of object migration in analyzing overload of load balancing and only consider the cost of migrating client connections. Third, most of these studies evaluated their algorithms through simulations instead of real virtual world systems.

The rest of the paper is organized as follows. In Section II, we analyze the benefits and overheads of distributed BSP. In Section III, we discuss approaches to optimize the workload migration process. In Section IV, we discuss the limitations of distributed BSP, the limitations of the simulator-centric architecture of virtual worlds, and present distributed scene graph as a new research direction. We then conclude in Section V.

II. Load Balancing Based on Distriuted BSP

Distributed binary space partitioning (BSP), especially partitioning a region with even bisectors, has the advantage of simple operations and local decision making, where regions can be split and merged quickly without global knowledge and expensive computations to determine the best partition plan. It has been compared with several other dynamic partitioning methods through simulations and shown to outperform other space partitioning methods in terms of inter-region communication overheads [10].

In this section, we study the effectiveness and limitations of distributed BSP in the context of virtual worlds. Our goal is to identify fundamental limitations and performance tradeoffs with real scenarios running on live servers. We use the following terminology in our discussions.

- Region: a piece of space in a virtual world.
- Prim: a primitive object, for instance, a cube or a sphere. It is the basic building block for composing more complex objects.
- Object: a set of prims linked together to represent a logical object in world.
- Source region: the region migrating workload away.
- Migrating space: the portion of the space from which workload is migrating away.
- Receiving region: the region that receives workload.

A. BSP Prototype

We have implemented our distributed BSP prototype based on OpenSimulator (abbreviated OpenSim) version 0.6.6. OpenSim adopts a system architecture similar to Second Life, as shown in Figure 1. Its core is a set of simulators. Each simulator hosts one or more regions and is responsible for the complete computation and communication work in the regions. (OpenSim follows Second Life's convention and statically partitions a world into 256m x 256m regions.) There are also infrastructure services for user authentication and authorization, asset and inventory service, messaging service, and grid service for assisting inter-region communications. These services are usually separated from the core simulations and hence are out of the scope of this paper.

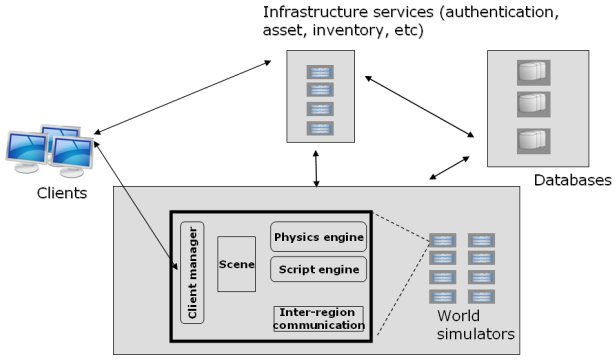

Figure 1 Simulator-centric architecture

In OpenSim, each region is an independent unit for virtual world simulation. Its runtime has several major components. The Scene contains the region's information, the objects inside the region, and a set of region modules for operations on the Scene. Physics Engine provides physics simulation in the region. Script Engine executes scripts to simulate object behaviors. Client Manager manages connection with clients and delivers world updates. Finally, Inter-region Communication provides communication to other regions and infrastructure services.

In our distributed BSP prototype, as shown in Figure 2, we extended the Scene's functionality and Grid Service to support rectangular regions with arbitrary sizes. In addition, we implemented a set of modules to enable dynamic space partitioning and migration of massive number of objects. The Partitioning Policy module implements a policy to decide when to split or merge workload. The Dynamic Partitioning

Region module (DPR) manages processes of splitting, merging and workload migration. The Dynamic Partitioning Engine (DPE) is responsible for inter-simulator communications to negotiate workload migration. The Resource Monitor is a global service and keeps tracks of simulator load status and responds to resource queries (e.g. idle simulators). In our experiments, it is also used to enable and disable dynamic partitioning to control the experiments.

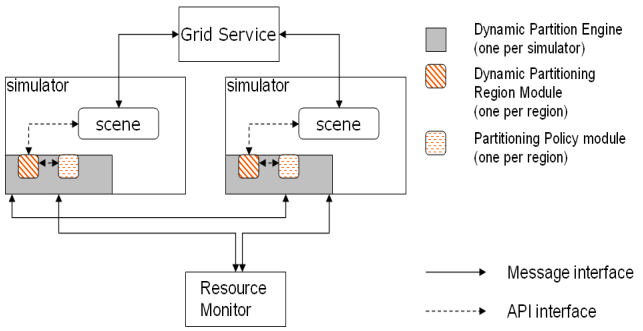

Figure 2 Components implemented or enhanced in OpenSim to support distributed BSP

Distributed BSP algorithm Our implementation of distributed BSP works as follows. (In our experiments, we used a simple load balancing policy based on the total number prims in a region to estimate workloads.)

1. Region split: A region is partitioned when its DPR detects that its load has exceeded a high watermark. Its DPE then queries the Resource Monitor for an idle simulator and migrates the workload in half of the space to the idle simulator.
2. Region merge: When a region's DPR detects that the region's load has been below a low watermark, it negotiates with its neighboring regions to merge. Two regions can merge only if the combined space is still a rectangle and the combined load does not exceed ¾ of the high watermark [10].

Workload migration There are different ways to migrate workloads with different performance. We started with a simple "freeze and migrate" implementation. Optimizations of the migration process are discussed in Section III.

1. For all the objects that have not been migrated in the migrating space, lock it if it is not being operated by any actor (e.g. script engine). Otherwise, mark it as "to-be-migrated" to prevent it to be operated again by any actor once the current operation is done.
2. Serialize the objects collected in step 1 and send to the receiving region. If the transmission is successful, the objects are removed from the source region.
3. Repeat the above steps until all objects in the migrating space have been migrated.

B. Evaluation Methodology

To thoroughly evaluate distributed BSP, we conducted our experiments with two types of workloads: real workloads and synthetic workloads.

The real workloads were collected from ScienceSim [7] regions, where ScienceSim developers and users have collectively developed significant amount of content. For instance, the Chamomile region used in our experiments was from a version in August 2009 and has more than 120K prims. On the other hand, most of these prims are static, such as bricks that constitute a building, and the number of prims that move around is relatively small.

We also generated a set of synthetic workloads to emulate regions with high dynamics. Avatars might be the most dynamic objects in virtual worlds. They may wander around, gather in one place and make some movements here and there, or move towards a hotspot. In addition, each avatar may have many prims attached to it and move with it. Hence we used the Random Waypoint Mobility Model [17], a model widely used in literature to model avatar movements, to generate movement patterns. The basic model works as follows: For a given space, each object randomly picks a destination (waypoint) and moves to it at a random speed uniformly chosen from 0 to *max_speed*. Once arriving at the waypoint, the object sleeps for a random amount of time bounded by *max_sleep*. Then it chooses its next waypoint and continues. The workloads used in our experiments include:

- Skewed workloads: Given the size of a region as (X,Y), where X is the size on *x* axis and Y the size on *y* axis, the objects are equally divided into *n* groups. For group *i*, *i* = 1 to *n*, the waypoints of each object in the group are randomly chosen from (*i*X/*n*, *i*Y/*n*).
- Cluster workload: The objects are randomly divided into *n* groups. For each group, a bounding box is randomly chosen in the space, and the waypoints in that group are generated inside the bounding box.

Each workload contains a certain number of objects that are generated within an initial region of 256m x 256m. The initial state of a workload is saved in an oar file (OpenSim Archive) and loaded into a OpenSim simulator at the start of each experiment. Figure 3 shows our setup to automatically drive the synthetic workloads to repeat experiments under similar conditions. The waypoints of each object are stored by an external http server. Every object runs a script that communicates with the external server to fetch the waypoints at run-time, and periodically resets the object's position towards the next waypoint with the given speed. Finally, a TestClient [8] (a client program that can connect to OpenSim regions and execute basic bot controls) connects to the initial region and issues commands (using the "say" function) to instruct the objects to fetch waypoints and start movements after all objects have obtained waypoints. Dynamic partitioning is enabled after all objects have started moving.

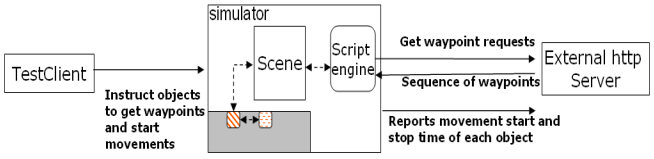

Figure 3 Experiment setup to drive dynamic workloads

C. Evaluation Metrics

We have instrumented the prototype and wrote scripts to collect the following metrics for evaluation. (For consistent setups, one simulator only hosted one region.)

- **Workload completion time:** In our workloads, the accumulated time of traveling on each leg and sleeping in between is the same for all objects. We define the workload completion time as the time from the first object starts moving to the time the last object stops moving. It measures a simulator's performance on executing scripts and processing events. The faster it executes, the more frequent it sends out updates and the better the user experiences.
- **Average CPU utilization** of each simulator.
- **Overload score**: In general, the score reflects the amount and degree of overloaded regions. This metric can have different definitions given different partitioning policies. For the policy based on prim count, this score is calculated as defined in [10]:

$$\theta = \sum_{k \in \{k_s > c\}} k \frac{k-c}{c}$$

where k_s is the number of prims in region s, and c is the maximum number of prims that can be supported by a region. (The calculation assumes one region per simulator and one simulator per server.)

- **Total number of migrated prims**: The total number of prims migrated from one region to another during the entire execution of a workload.
- **Object unavailable time during migration**: During migration, for synchronization purpose, an object's state is locked and it does not send out updates or respond to operations. Once it arrives at the new region and resumes execution, its state is unlocked and it is accessible again. The time in between is the object unavailable time. This metric directly affects user experiences and should be minimized.
- **Total number of region-crossing**: It measures that during the entire execution of a workload, the total number of prims handed over between regions due to object movements. When the space is partitioned to a set of smaller regions, an object could move to a position that is beyond the space of its current region and need to be transferred over to another region. We count each handover event as one region-crossing.

D. Results

We first present our results with synthetic workloads. In our experiments, we assume the available resources as a set of homogenous servers, that is, each of the servers can host up to N prims. The evaluations are focused on how BSP partitions the initial region for each workload and how much overhead is cost by dynamic partitioning.

In our experiments, the OpenSim simulators, the grid server, the Resource Monitor, and the external http server for waypoint inputs, all run on an Intel Xeon E5450 machine. It has two quard-core processors and 12G memory. The database runs on a different machine located in the same rack. Each experiment started with the initial region hosted by one simulator. In all workloads, the total time for each object to travel through all legs plus the sleeping time in between was set to 600 seconds. The values for *max_speed* and *max_sleep* were set to 10m/s and 6 seconds, respectively.

TABLE 1 EXPERIMENT SETUPS

Workloads	Setups	Each simulator's capacity	Total simulator capacity
Skewed: 80, 160, 320, and 640 prims, respectively	8-sim, lower capacity	15, 30, 60, and 120 prims, respectively	120, 240, 480, and 960 prims, respectively
	4-sim, higher capacity	30, 60, 120, and 240 prims, respectively	
Cluster: 80, 160, and 320 prims, respectively	8-sim, higher capacity	30, 60, and 120 prims, respectively	240, 480, and 960 prims, respectively
	16-sim, lower capacity	15, 30, and 60 primss, respectively	

We have generated synthetic workloads that each has 80, 160, 320 and 640 moving objects, where each object consists of only one prim. For each workload, we run experiments with two setups: a smaller number of simulators each with a higher capacity, and a larger number of simulators each with a lower capacity. Table 1 summarizes all the setups.

1) Effectiveness of distributed BSP

First, we study if distributed BSP is able to balance workload among the simulators when the total system capacity exceeds the workload. (For example, in Table 1, for a workload with 80 objects, the total simulator capacity was set to 120.) Figure 4 shows the region partitions from two experiments of the skewed workload with 640 objects, where the experiments have different setups but the same total system capacity. Figure 5 shows the overload score as dynamic partitioning progressed for the same experiments in Figure 4. With both setups, the initial region (the entire space as shown in each figure) was partitioned into smaller regions until none of the regions was overloaded (the overload score eventually went down to 0, as shown in Figure 5). The difference is that with the "8-sim, lower capacity" setup, the workload was distributed to more regions and it took longer to balance the load. In other words, for the same workload, smaller per server capacity resulted in more region partitions.

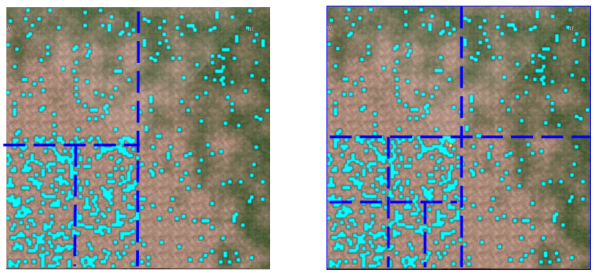

Figure 4 Results of region partitions with skewed workload, 640 prim (each dot represents a prim) (a) 4-sim, higher capacity, (b) 8-sim, lower capacity

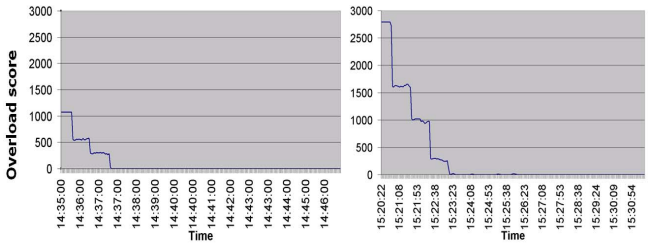

Figure 5 Overload score for results in Figure 4 (a) 4-sim, higher capacity, (b) 8-sim, lower capacity

Figure 6 presents the partitioning results for cluster workloads. With the "8-sim, higher capacity" setup, eventually workload was distributed to 6 regions. With the "16-sim, lower capacity" setup, the workload was distributed among 13 regions. Figure 7 shows the initial steps of region splitting and merging for the cluster workload with 80 objects. It shows that distributed BSP is able to dynamically split and merge regions to balance load.

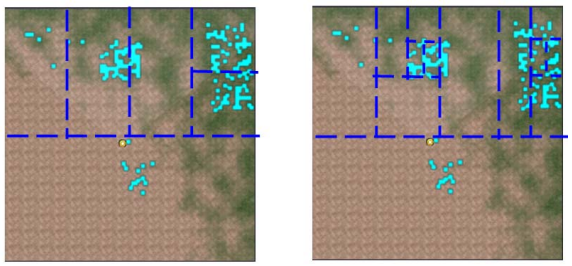

Figure 6 Region partitions from cluster workloads (a) 8-sim, higher capacity, (b) 16-sim, lower capacity

Note that for workloads with the same number of objects, the cluster workloads resulted in more region partitions than the skewed workloads. The reason is that in the skewed workloads, objects spread out across the space and the object density is more proportional to space.

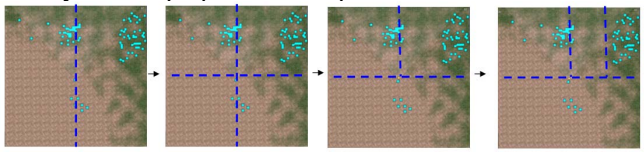

Figure 7 Initial steps of region splitting/merging, cluster waypoint workload, 80 prims.

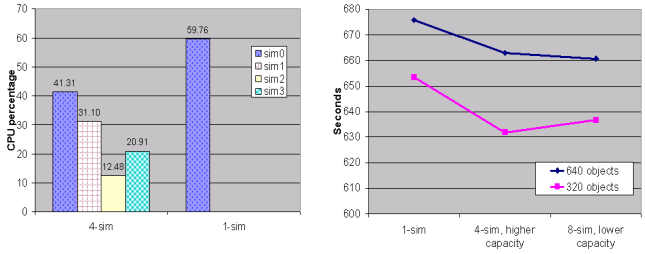

Figure 8 Skewed workload, 640 prims: (a) Average CPU utilization per simulator, (b) workload completion time comparison

The above results demonstrate one benefit of dynamic partitioning over static partitioning, that it is able to adapt to dynamic workloads and effectively distribute load among simulators. Another benefit of dynamic partitioning is its potential to scale user experiences with more available resources. Figure 8 shows the performance comparison of the skewed workload with 640 objects, when there are different numbers of simulators available. Figure 8(a) compares the average CPU usages of each simulator during the entire workload execution. The left bars show the results where the workload was eventually distributed among 4 simulators, while the right bar shows the CPU usage when the workload was run on only one simulator. In the case of 4 simulators, sim0 was the simulator hosting the initial region and gradually migrated workload to other simulators.[1] With more simulators to share load, the average CPU usage on sim0 is significantly reduced, demonstrating the potential of dynamic partitioning to alleviate hotspot problems by leveraging additional hardware. Figure 8(b) shows the workload completion times averaged over multiple runs. In our experiments, ideally, this completion time should be 600 seconds (recall that the time span for each object's movement in all synthetic workloads is 600 seconds).[2] In Figure 8(b), with the 640 objects workload, a single simulator was only able to finish the workload in 677 seconds. By distributing workload to more simulators, the completion time was reduced to about 661 seconds (for both 4-sim and 8-sim setups), even though there were extra delays added by workload migration.

2) Overheads

Overhead of region-crossing Results in the previous section show that for workloads with disproportional object density (e.g. the cluster workloads), or systems with smaller per server capacity, the initial region was eventually split into more region partitions. Intuitively, the more region partitions, the higher the probability that objects need to be handed over across regions when they move around, and hence the higher the overhead. Figure 9 shows the results of region-crossings from all skewed workloads. (Similar trend was observed in cluster workloads.) Each data point was averaged through multiple runs, though the differences between different runs were actually small. Note that for the same workload, the total system capacities of the setups of "8-sim, lower capacity" and "4-sim, higher capacity" are the same (see Table 1). Figure 9(a) presents the total number of region-crossings in each workload. With the same total system capacity, the "8-sim, lower capacity" setup had larger number of region-crossings. This indicates that with smaller capacity servers, a larger portion of the system capacity has to be devoted to handle region-crossing, hence making the system more prone to workload increases. To better illustrate this point, we plot the ratio of the total number of region crossings over the total system capacity in Figure 9(b). It

[1] Besides migration overheads, the CPU usages on each simulator in the 4-sim case also include the overheads such as to create new regions and load executable modules for the new regions, for which we just used the implementation in OpenSim. These overheads could be optimized, for example, by pre-creating empty regions.

[2] For those who are familiar with OpenSim's configuration file, we set ScriptDelayFactor to 0, so that each object's script that move itself around will immediately return after it sets the object to a new position.

shows that the ratio for "8-sim, lower capacity" is almost twice of the ratio for "4-sim, higher capacity".

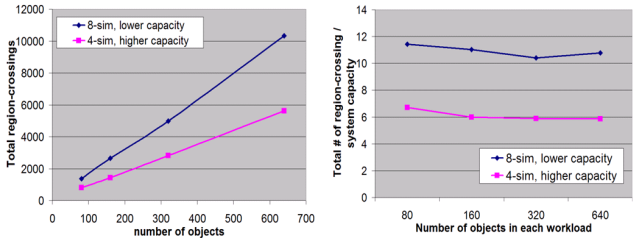

Figure 9 Region-crossings from skewed workloads (a) total number of region-crossings, b) ratio of region-crossings to system capacity

Overheads from workload migration Figure 10(a) presents the total number of migrated prims from all cluster workloads. For the same workload, the total system capacities of the setups of "16-sim, lower capacity" and "8-sim, higher capacity" are the same. Again, we see that the "lower server capacity" setups experience higher number of total prim migrations. With lower capacity servers, the initial region had to be divided into more smaller regions and hence more prims were migrated from one region to another. Figure 10(b) plots the ratio of the total number of migrated prims over the total system capacity. Similarly, it indicates that with the same total system capacity, systems with "lower server capacity" would have to spend a higher portion of system capacity to handle workload migrations.

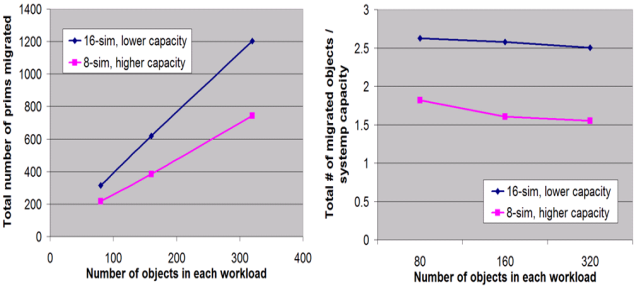

Figure 10 Total number of prims migrated in cluster workloads (a) total number of migrated prims, (b) ratio of migrated prims vs. system capacity

Next, we study the object unavailable time during migrations. Our initial implementation of the migration process was described in Section II.A. To get a sense of the worst case object unavailable time (for instance, the server hosting the source region is heavily overloaded and cannot effectively parallelize the migration tasks) as a baseline for comparison, we implemented the migration process sequentially. The source region first froze objects in the migrating space, then it sequentially serialized these objects. Next it transferred the serialized data to the receiving region, and the receiving region deserialized the data and created the objects sequentially. We used OpenSim's existing implementation for serialization and deserialization, which is based on XML formats. When an object was created on the receiving region, all of its scripts were created and started execution from the "frozen state". The script creation (called "rezzing scripts") in OpenSim was implemented asynchronously and hence run concurrently.

Figure 11(a) shows the average object unavailable time in each migration, ordered by the number of migrated prims, and Figure 11(b) presents the amount of serialized data transmitted between regions in the migration processes. The results are from all synthetic workloads with all setups. The unavailable time of an object was measured from the time when the state of all objects in migration was frozen, to the time that the object has been fully deserialized and its scripts have all resumed. The trend suggests that in worst cases (things happen sequentially), migrating 100K such prims could take thousands of seconds. That would be intolerable user experiences. The large delay is mainly due to serialization of a large volume of objects. Note that although each object only had one prim, it had a non-trivial script attached. During migration, the script's state and its assembly code (to save compilation time at the receiving region) were also serialized. As a result, each object's serialized data was as big as about 50KB, and more than 10MB of data were transmitted in migrating 200 objects.

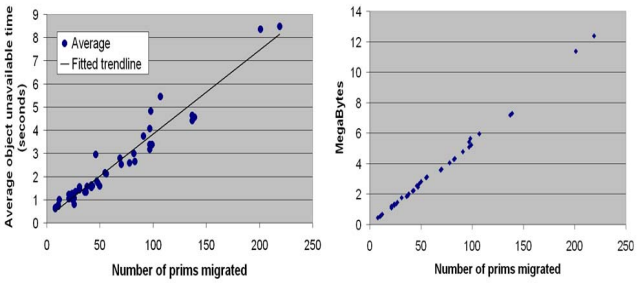

Figure 11 Migration overhead, ordered by the number of migrated prims (a) object unavailable time (b) amount of data transmitted in each migration

The results reveal that object migration in virtual worlds is an expensive process and should be carefully optimized. Another observation is that for workload with high dynamics, it is beneficial to use a set of large-capacity servers to reduce the overheads of inter-region communications and object migration.

III. Optimization of Object Migration

With complex scenes in virtual worlds, it is possible that millions of prims need to be migrated in load balancing processes. It is critical to optimize the migration process and reduce object unavailable times to an acceptable range.

There could be different ways to optimize the process, which can be applied as supplements to each other. The first one is to parallelize and streamline the tasks, for example, by creating multiple threads that each handles a portion of the objects. This method alone, however, can only reduce the object unavailable time linearly, approximately by the number of available hardware threads (if there is enough network bandwidth to support the concurrent traffic).

A second approach is to apply more efficient serialization and deserialization methods. Using XML strings to represent objects, as in current OpenSim, results in large amount of data to be serialized and transferred. It could be optimized or replaced by other methods.

A third approach is to apply a "pre-copy" phase, as used in virtual machine migrations [16]. Such an approach includes two phases, "pre-copy" and "stop-and-sync". In the "pre-copy" phase, objects in the migrating space are serialized and transferred to the receiving region, with the execution continues on the source region. In other words, these objects are not frozen during the "pre-copy" phase. Only after they have all been copied to the receiving region, will they be frozen and the second phase, "stop-and-sync", start. In the second phase, for any object in the migrating space whose state has changed since the "pre-copy" phase, its state will be synchronized with the receiving region. Objects are only unavailable during the stop-and-sync phase. Further, since only updates of the state, instead of the entire state of an object, need to be transferred in this phase, the object unavailable time could be dramatically reduced. This approach trades off system performance with object unavailable time: the source region needs to continue execution on the migrating space during the pre-copy phase.

One important observation from the traditional architecture of virtual worlds (Figure 1), is that there usually is a centralized database (central in terms of administration domain) to store assets and to persist the world's state. Virtual world simulators periodically synchronize with the database for persistence and failure recovery purposes. This observation brings the idea of leveraging the objects already stored and constantly updated in the database to enhance the "pre-copy" phase, referred as "migration with database support". The basic algorithm works as follows.

1. The source region starts the pre-copy phase and goes through all objects in the migrating space. If the state of an object has changed since last time the region synchronized with databases, the current state is saved to databases. The source region then informs the receiving region to start loading the objects. Meanwhile, the source region continues the execution of all objects it is hosting.
2. The receiving region loads the objects in the migrating space from databases and informs the source region when the loading completes.
3. The source region starts the "stop-and-sync" process. It freezes the objects in the migrating space and transfers state updates since the "pre-copy" phase, if any, to the receiving region.
4. The receiving region applies all the updates and informs the source region to remove the migrating space and the objects in the space.

We enhanced our prototype with the above algorithm, and applied it to dynamically partitioning workloads from ScienceSim regions. Figure 12(a) shows a snapshot of the Shengri La Chamomile region as of August 2009. It has 124,583 prims. These prims were composed into objects such as waterlilies, rotating swans, a big building, and furniture in and outside the building. Most of the prims were concentrated in the right-bottom quarter where the building located. In the experiments, we used a setup that assumed each server's capacity was 40K prims. Figure 12(b) shows the final region partitions. Eventually the initial region was split into 8 partitions and 7 workload migrations happened.

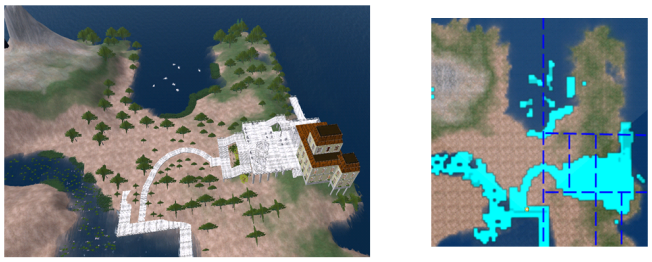

Figure 12 The Chamomile region in ScienceSim (a) a snapshot of the scene (2) regions partitions by applying BSP, assuming each server's capacity is 40K prims.

For each migration process, Table 2 presents the number of migrated prims and the delays in each migration phase. In such real workloads, a large portion of objects are static and rarely change states. (The same observation was made in a Second Life analysis [27]). Hence not too many objects need to be synchronized with the database during "pre-copy". As shown in Table 2, the "pre-copy" phase was completed within 1 minute in all migrations. The "object unavailable times", which are the same with the delays in "stop-and-sync", were even smaller. As shown in the first row of Table 2, migration of more than 120K prims resulted in less than 35 seconds of the unavailable time. This is orders of magnitude reduction from previous results of thousands of seconds. Note that for fair comparison, the "stop-and-sync" phase still used the implementation of "sequential serialization and deserialization" as discussed before. The optimizations discussed earlier, namely multi-threading and efficient serialization methods, can be applied in this phase to further reduce delays.

TABLE 2 OBJECT MIGRATION DELAYS FROM PARTITIONING THE CHAMOMILE REGION

Total number of migrated prims (and the corresponding number of objects)	Delay in pre-copy (seconds)	Delay in stop-and-sync (seconds)	No. of prims sync'ed in stop-and-sync phase
120538 (5472 objects)	60.3	34.5	3695 (40 objects)
1135 (136 objects)	7.36	2.82	328 (5 objects)
70806 (3956 objects)	38.65	16.94	2344 (24 objects)
45225 (1084 objects)	33.97	6.7	930 (10 objects)
25355 (1790 objects)	58.74	14.67	2250 (22 objects)
32508 (933 objects)	34.1	6.64	930 (10 objects)
13231 (698 objects)	21.31	0	0

The objects involved in the "stop-and-sync" phase are mainly the objects that frequently change states, such as rotating swans and ceiling fans that have scripts attached. In the current version of OpenSim, scripts are stored separately in each simulator's local directories. As an enhancement, they could be stored in databases or shared file storages, which could then further reduce the data to be transferred in the "stop-and-sync" phase and consequently, the delays in this phase. We are working on this optimization to further reduce object unavailable times.

IV. DISCUSSIONS

We initially chose to study distributed BSP because it is simple and does not require global optimization algorithms. Our evaluations, however, reveal that although it is effective in balancing workload dynamically, it has several limitations and suffers from high overhead. This leads us to the thoughts of the requirements of load balancing in virtual worlds, the fundamental reasons of the observed limitations, and how to address the limitations.

A. Requirements of load balancing in virtual worlds

In general, the goal of load balancing in virtual world is to satisfy user experiences with the least cost. It needs to maintain the experience of real-time interaction and visualization, and optimize operations for a set of heterogeneous actors. Hence, a load balancing algorithm needs to have the following properties.

- Flexible configuration for scalability: It should enable virtual world to scale flexibly by dynamically adding resources to match load, especially when workload changes dramatically (such as the crowding [12][13] or flocking [20] scenarios).
- Low overhead: It should have unnoticeable service interruption time while migrating workload and minimize inter-server communications caused by load balancing.
- Optimal usage of resources: First, the algorithm supports fine-grained workload adjustment to flexibly match load with server capacity. Second, it is able to match actor operations with appropriate hardware.

B. Limitations of Distributed BSP

Distributed BSP supports flexible configuration for scalability. Our results demonstrate that when there is additional resource available, distributed BSP is able to partition workload and alleviate the hotspot problem. On the other hand, the results also revealed that it suffers from high overhead and has limitations on optimal usage of resources. We observe that the main reasons are region fragmentation, lack of fine-grained workload partitioning, and the homogenous simulator-centric system architecture. Here we focus our discussion on the first two reasons and will discuss the third one in the next sub-section.

Distributed BSP suffers from region fragmentation due to its limitation on merging regions. To preserve simplicity, each region maintains a regular shape. Two adjacent regions can only be merged if their union is also the same regular shape. In other words, in many cases, adjacent regions cannot be merged. This fragmentation of partitioned regions produces many under-loaded regions and also incurs high overhead of inter-region communication. The problem becomes more serious when workload tends to concentrate in small areas or each server has relatively small capacity.

Region fragmentation also contributes to the high migration overhead. When overloaded, a server simply splits a region and sheds away half of the space and the workload in it to another server (hence the "coarse-grained" workload partitioning). It might take several rounds of region splitting or merging to balance load. As a result, many objects may be migrated multiple times during the process.

To reduce the migration overhead due to region fragmentation, two approaches can be applied: 1) pre-partitioning the space with recursive BSP before migrating any workload, 2) micro-cell partitioning.

Recursive BSP has been applied to produce the initial partitions and gradually refine the partitions, where each partition is a union of regular shape regions [18]. Some approaches also apply techniques such as space filling curves to produce partitions and minimize disjoint regions in each partition [19]. Existing solutions, however, are designed as centralized algorithms that require global knowledge and expensive computations to decide the best partition plan.

In micro-cell approaches, a virtual space is divided into small cells [20][21][22][23][24]. Cells are grouped into partitions and each server hosts one partition. A cell is the base unit for neighboring partitions to negotiate and migrate workloads. This is a promising direction although most existing studies focused only on load balancing client connections and ignore the load of other actors. Analysis of the overhead of object migration has also been largely overlooked.

Finally, a general problem of space based partitioning is that there is no consideration of logical connections between objects. For instance, partitioning may cut a room in half when it might have been more efficient to keep the room as a whole for physics simulations. Enhancements such as using meta-data to describe logical connections among objects might be applied to address this problem.

C. Limitations of Simulation-centric System Architecture

Existing studies of virtual worlds, including our evaluation work in previous sections, are almost all based on the simulator-centric architecture, as shown in Figure 1. In this architecture, the scene is an internal component in a simulator and is tightly bounded to all the actors operating on it such that the scene and the actors are physically co-located. We have observed several limitations of this simulator-centric architecture.

1) High overhead of workload migration

Our results showed that workload migration is a significant performance factor in load balancing and should be minimized as much as possible. With the simulator-centric architecture, however, every time a region is split or merged, the objects in the migrating space as well as the ownership of all simulations in that space needs to be transferred from one simulator to another. It means that besides object migration, the operations of all actors must be reloaded in the receiving simulator, which is also expensive (for instance, some actors need to create their own internal representation of objects for operation purpose when they are loaded).

2) Inefficient partitions for some actors

Another important observation is that when a simulator is overloaded, usually it is one actor that is overwhelmed by the volume of objects it operates on or by the scope of its

operations. Then to balance load, the region has to be partitioned. Although the partitioning is necessary for the overwhelmed actor, it may not be necessary for other actors operating on the same region. Moreover, different actors may prefer different policies to partition workload, which could be conflicting. For instance, physics engine may prefer partitions that preserve locality, while client managers may prefer partitions based on area-of-interest of clients. To understand this point better, we next discuss the actors in more details. Generally, actors that operate in a virtual world include, but not limited to:

- client manager, which manipulates the world on behalf of a client and forwards updates of the world's state to clients;
- per object actors, such as running scripts of an object that defines the object's behaviors;
- small-range actors operating on a limited number of objects, such as collision detection in physics engines;
- broad-range actors operating on a large number of objects, such as N-body interactions or protein folding in scientific simulations.

These actors vary by the following factors:

- scope of operations, for instance, how many objects are affected by an actor's operation,
- performance constraints, such as (1) latency constraints (responsiveness) and (2) bandwidth constraints,
- complexity and characteristics of operations [14].

Given the above differences, it is hard to apply a general partitioning strategy that satisfies the requirements or constraints of all actors. In fact, most existing research has mainly focused on satisfying the requirements and constraints of client managers and few have taken other actors into consideration.

3) Limited ability to apply appropriate hardware to heterogeneous actors

Different actors have different computation or communication characteristics. For instance, our measurements show that script engine has scattered memory access while physics engine has high locality of memory access [14]. Yet aggregating the heterogeneous actors in homogeneous simulators limits a virtual world's ability to apply "appropriately configured" hardware to support different actors.

D. New architecture and open research questions

By observing the limitations of the simulator-centric architecture, we argue that a different architecture is needed to break the aggregation of actor operations in homogenous simulators. We are currently working on an architecture called Distributed Scene Graph (DSG) [9] [14], as illustrated in Figure 13.

DSG views the virtual world operations in general as a collection of the "Scene" and the actors operating on the Scene through a scene service layer. An actor may or may not locate in the same machine with the portion of the scene it operates on. By detaching actors from the Scene, DSG enables running actors on separate hardware and applying specialized optimizations. On the other hand, Scene could be ignorant of operations or existence of specific actors and mainly focus on data management, state synchronization, and event distribution. A more detailed discussion of DSG can be found at [14].

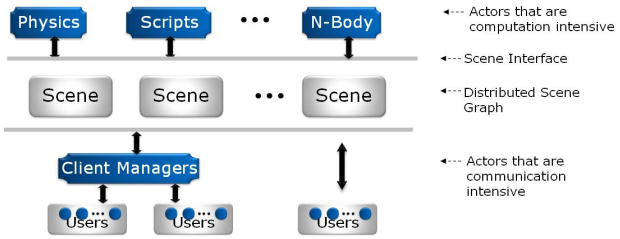

Figure 13 Distributed Scene Graph (DSG)

There are a number of research questions for load balancing under the DSG architecture. Addressing these research challenges is of our future research:

- algorithms to effectively distribute Scene and actors to appropriate hardware and adjust the operation scope for each actor, with minimized service interruption,
- metrics for evaluating the effectiveness and overheads of load balancing solutions,
- synchronization protocols between the Scene and the actors to achieve desired data consistency with minimized overhead.

In related works, the architecture design in Project Darkstar [25] has a similar spirit with DSG in that it detaches tasks (a set of action in response to messages or events) from data and maintains the data in a data store. As an alternative, Meru [26] proposes to break a virtual world into space servers, object hosts, and a resource content distribution network. All these approaches are initial steps toward re-architecting virtual worlds for scalability.

V. CONCLUSIONS

Dynamic load balancing holds the potential to scale virtual worlds flexibly by dynamic allocation of resources to match dynamic workload. Our work revealed that although being simple and effective, distributed BSP suffers from region fragmentation and high overhead of workload migration and inter-region communication. Further, we analyzed the simulator-centric architecture used in today's virtual worlds and showed that it has fundamental limitations on reducing load balancing overhead and leveraging hardware efficiently.

We believe that to broaden the solution space and inspire new approaches to scale virtual worlds, it is necessary to break away from the simulator-centric architecture that aggregates operations of all the actors. Currently, we are working on load balancing solutions that are based on a new architecture called Distributed Scene Graph. There are still many open research problems that call for innovative solutions.

ACKNOWLEDGMENT

We thank Robert Adams, John Hurliman, and Dan Lake from Intel Labs for their help in developing the prototype and conducting experiments, and for their valuable feedbacks to the paper.

REFERENCES

[1] Interview from virtualworldsnews, http://www.virtualworldsnews.com/2008/06/strategy-analyt.html

[2] Second Life®, http://www.secondlife.com

[3] World of Warcraft™, http://www.worldofwarcraft.com

[4] OpenSim, http://opensimulator.org/wiki/Main_Page

[5] OpenSim Grid Architecture, http://opensimulator.org/wiki/Grid_Architecture_Diagram

[6] Second Life Architecture, http://www.infoq.com/news/2008/12/Second-Life-Ian-Wilkes

[7] ScienceSim, http://www.sciencesim.com/wiki/doku.php

[8] TestClient, http://lib.openmetaverse.org/wiki/TestClient

[9] C. M. Bowman, D. Lake, and J. Hurliman. Designing Extensible and Scalable Virtual World Platforms. *Extensible Virtual Worlds Workshop (X10)*, 2010.

[10] F. Chang, C. M. Bowman, W. Feng, XPU: A distributed architecture for metaverses. Technical Report 10-04, Department of Computer Science, Portland State University, 2010.

[11] N. Gupta, A. J. Demers, J. Gehrke, P. Unterbrunner, and W. M. White: Scalability for Virtual Worlds. In *Proceedings of the 2009 IEEE International Conference on Data Engineering* (ICDE 2009).

[12] I. Kazem, D. T. Ahmed, and S. Shirmohammadi. A Visibility-Driven Approach to Managing Interest in Distributed Simulations with Dynamic Load Balancing. In *Proceedings of the 11th IEEE international Symposium on Distributed Simulation and Real-Time Applications* (DS-RT), 2007.

[13] F. Lu, S. Parkin, and G. Morgan. Load balancing for massively multiplayer online games. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support For Games* (NetGames) 2006.

[14] H. Liu, M. Bowman, R. Adams, J. Hurliman, and D. Lake. Scaling virtual worlds: simulation requirements and challenges. In *Proc. of Winter Simulation Conference*, 2010, in press.

[15] P. Morillo, J. M. Orduna, M. Fernandez, and J. Duato. Improving the Performance of Distributed Virtual Environment Systems. *IEEE Transactions on Parallel and Distributed Systems archive*, Vol. 16 , Issue 7, July 2005.

[16] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In Proceedings of the *2nd conference on Symposium on Networked Systems Design & Implementation*, 2005.

[17] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, volume 353, pages 153-181. Kluwer Academic Publishers, 1996.

[18] J. C. S. Lui and M. F. Chan. An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems. *IEEE Trans. Parallel Distributed System*, Vol. 13, No. 1, 2002.

[19] P. C. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco. Dynamic Octree Load Balancing Using Space-Filling Curves. *Williams College Department of Computer Science Technical Report CS-03-01*, 2003.

[20] J. Chen, B. Wu, M. Delap, B. Knutsson, H. Lu, and C. Amza, Locality aware dynamic load management for massively multiplayer games. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, 2005.

[21] K. Lee and D. Lee. A scalable dynamic load distribution scheme for multi-server distributed virtual environment. In *Proceedings of the ACM symposium on Virtual reality software and technology (VRST)*, 2003.

[22] D. T. Ahmed and S. Shirmohammadi. A Microcell Oriented Load Balancing Model for Collaborative Virtual Environments. In *Proceedings of IEEE International Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems (VECIMS)*, 2008.

[23] B. De Vleeschauwer, B. Van Den Bossche, T. Verdickt, F. De Turck, B. Dhoedt, and P. Demeester. Dynamic microcell assignment for massively multiplayer online gaming. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support For Games* (NetGames), 2005.

[24] T. N. B. Duong and S. Zhou. A Dynamic Load Sharing Algorithm for Massively Multiplayer Online Games. In *Proc. of 11th IEEE International Conference on Networks* (ICON), 2003.

[25] J. Waldo. Scaling in games and virtual worlds. *Communications of ACM* 51 (8), 38-44, 2008.

[26] D. Horn, E. Cheslack-Postava, B. F.T. Mistree, T. Azim, J. Terrace, M. J. Freedman, and P. Levis. To Infinity and Not Beyond: Scaling Communication in Virtual Worlds with Meru. Technical Report CSTR 2010-01, Dept. of Computer Science, Stanford University, 2010.

[27] M. Varvello, F. Picconi, C. Diot, E. W. Biersack. Is there life in Second Life? In *Proc. of 4th ACM International Conference on emerging Networking Experiments and Technologies* (CoNEXT), 2008.