Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

ENABLING FINE-GRAINED LOAD BALANCING FOR VIRTUAL WORLDS WITH DISTRIBUTED SIMULATION ENGINES

Arthur Valadares
Cristina Videira Lopes

Donald Bren School of ICS
University of California, Irvine
6210 Donald Bren Hall
Irvine, CA 92697-3425, USA

Huaiyu Liu

Intel Labs, Intel Corporation
2111 NE 25th Ave
Hillsboro, OR 97124, USA

ABSTRACT

Virtual worlds are general-purpose real-time simulation of three-dimensional environments, and serve for several purposes, such as physics simulation, collaboration, and entertainment. Due to the real-time nature of these simulations, scaling the number of in-world entities and interacting users is challenging. In this paper we present a novel approach to scalable virtual worlds, combining two dimensions of workload partitioning: space and operations. We present this new design as the Distributed Scene Graph with microcells (DSG-M), and evaluate our approach in a distributed physics intensive evaluation aimed at testing two hypothesis: (1) the space partitioning approach improves scalability by balancing the load of an overwhelmed physics dedicated simulator; and (2) simulation precision can be maintained by assigning read-only spaces near the partition borders. Results show evidence to confirm both hypotheses, and of successfully scaling the simulation of an overwhelming workload.

1 INTRODUCTION

Virtual worlds are multi-purpose three-dimensional environment simulations, serving as a foundation for a diversity of applications, such as massively multiplayer online games (MMOG), collaborative environments (virtual meetings, events), and real-world simulations. Although all virtual world applications share the same foundational requirements of a virtual 3D world simulation, each application presents different priorities and challenges with regards to scalability.

There are multiple aspects to scaling a virtual world. Scalability may involve increasing the virtual land space, accommodating massive number of objects, high fidelity user interaction, realistic physics, or any combination of these aspects. Any solutions for virtual worlds should be flexible in addressing different workloads effectively. In previous work (**?**), we discussed the challenges of scaling virtual worlds with diverse workloads through space partitioning alone. We found that while space partitioning is effective, there are drawbacks due to high load of migrating workloads and communication overhead. Next, we proposed and evaluated a software architecture based on partitioning the operations of the virtual world, the Distributed Scene Graph (DSG) (**?**, **?**). The DSG architecture was shown to improve client connection scalability by a factor of 10.

In this paper, we further improve the scalability capacity of the DSG architecture by including a fine-grained space partitioning mechanism called microcells. DSG with microcells, or DSG-M, can improve DSG by enhancing load balancing to both operation and space partitioning simultaneously. In the evaluation, we will show that even simulations with strong space locality can be partitioned and distributed to scale horizontally. We also show that simulation precision is maintained, by introducing a read-only buffer

space around the partition border. Our technology is implemented as an OpenSimulator module, and is open-sourced and available at https://github.com/intelvwi/DSG.

This paper is organized as follows: in section 2, we present the necessary background on space partitioning, microcells, and the DSG architecture. In section 3, we show how microcells can be integrated to the design of DSG, using flexible space partitioning to reduce state propagation messages across simulators. In section 4, we evaluate our approach with a load-intensive physics experiment, and in section 5, we present our conclusions.

2 BACKGROUND

The most ubiquitous approach to scaling virtual worlds is the use of space partitioning algorithms. In space partitioning, the land space of the virtual environment is divided, and each partition is delegated to different servers who will be responsible for processing events, storing data, and treating client requests. There are two dimensions to space partitioning: (1) fixed-size and flexible; and (2) static and dynamic.

In fixed-size space partitioning (**?**, **?**), space is divided in equally-shaped spaces. Flexible space partitioning divides the world in partitions of different shapes and sizes. **?** uses small fixed size spaces called microcells to create flexible partitions. Another approach is using Voronoi diagrams (**?**) to partition space into cells, that can be joined to form different shaped partitions (**?**, **?**).

When the partitions are determined before execution, the partitioning is static. In dynamic space partitioning, or DSP, the partitions shape, size, and ownership can change at runtime. The Pikko Server (**?**) and EVE Online (**?**) are examples where space is partitioned based on runtime load balancing policies.

Some solutions approach scalability from an architectural perspective. The OpenWonderland virtual world (**?**) uses a multi-tier architecture called Darkstar. In Darkstar, multiple nodes are instantiated to balance the load. Each node is identical, containing the game server and the Darkstar stack. When an update is processed by the game server, Darkstar creates a transaction and requests the necessary data through a meta-service connected to the database. If the transaction cannot be completed due to conflicts, Darkstar aborts and retries when the data is available for writing.

In Meru (**?**), object execution and world simulation are separated. Objects are executed independently in object hosts, and the simulators, named space servers, perform object lookup and routing of object updates to clients. Space servers partitions the world by space, but are responsible for event routing, while object hosts are responsible for processing object updates.

Data Distribution Management (DDM) specifies distribution of data in High Level Architecture (HLA) compliant simulations (**?**). There are several DDM approaches for *interest management* (**?**), specifying different mechanisms for matching, filtering, and delivering messages efficiently to interested parties. Our space partition approach is similar to the dynamic grid-based approach, except we do not assume a network topology (e.g. multicasting groups). Designing an efficient network topology is paramount for scalability, but is out of the scope of this paper.

In previous work (**?**), we identified scalability challenges and solutions for partitioning virtual worlds without loss of consistency. In this section, we revisit the concept of space partitioning, and explain our previous work on the DSG architecture.

2.1 Space partitioning and Microcells

Traditional fixed-size space partitioning allows virtual environments to expand beyond the processing power of a single server, but each individual region is vulnerable to overload: if too many users or entities aggregate inside one region space, the server handling that region will perform poorly or crash.

Instead of partitioning with a coarse-grained fixed-size space, a finer-grained flexible partitioning is better adapt to the load created by virtual worlds. One approach to the flexible partitioning of virtual worlds is *Microcell partitioning* (**?**). Microcell partitioning improves the traditional fixed-size space partitioning approach. Microcells are small and indivisible areas of space that can be grouped in sets, forming partitions

of flexible size and shape. It is then possible to balance the load on servers by swapping authority over microcells as servers become overwhelmed.

2.2 Distributed Scene Graph

The Distributed Scene Graph (DSG) architecture distinguishes two independent concerns in virtual world simulations, data (*Scene*) and *operations*. *Operations* are organized by functional features into *actors*.

- **Scene**: Represents the data structure and state of the virtual world and its entities.
- **Operations**: Reads and writes the data structures and state of the Scene. Operations modify the Scene to simulate the virtual world, and are aggregated in functional features.
- **Actors**: Operations can be aggregated by functionality. For instance, physics operations are responsible for updating the Scene with physics-related behaviors such as collisions and acceleration. Script operations would update the same Scene, but concerned with scripted behaviors, such as updating an object size or location. These aggregated functional features can be consolidated in independent simulators, which we refer to as *actors*.

To provide a scalable virtual world, the workload should be partitioned and mapped to multiple servers when the load exceeds the capacity of a single server. Traditional workload partitioning work (**?**, **?**, **?**, **?**, **?**, **?**, **?**) focused only on Scene partitioning, leaving operations unpartitioned. We identified limitations of Scene load partitioning (**?**, **?**) due to heterogeneity of operations. Hence we enabled operation disaggregation in DSG so different simulators operating on the same Scene are capable of running on separate hardware and be load balanced independently.

DSG has been shown to improve scalability with actor partitioning (**?**), but partitioning by operations alone is not complete for fully enabling load balancing. Scene partitioning is also required when an actor becomes overloaded and needs to shed more load to available hardware resources. In the next section, we integrate microcell partitioning to DSG, so actors can also be partitioned in terms of space.

3 DSG with Microcells (DSG-M)

DSG-M results from the combination of DSG and Microcells, i.e. ”operations” partitioning as well as flexible space partitioning. DSG enables workload to be partitioned in actors, and actors can be instantiated many times for improved load balancing. For instance, a client manager actor, responsible for receiving and sending updates to clients, could be instantiated several times to handle more clients. In DSG-M, workloads can be addressed by space partitions as well. Physics actors, responsible for the physics operations, can be instantiated and assigned to smaller parts of the total workload, by being assigned smaller areas of space. Our work adapts microcells to DSG, enhancing load balancing through flexible space partitioning, in parallel to operation partitioning. We refer to our adapted architecture as DSG-Microcell, or DSG-M.

Integrating DSG with microcells poses some challenges. The original concept of microcells is constrained to space partitioning only, with no disaggregation of operations. This way each partition has only one authoritative simulator capable of performing write operations. By merging operation partitioning, there is no longer a single authority for each entity. It becomes possible for multiple actors to perform conflicting updates on the Scene, requiring additional constraints for maintaining eventual consistency. Thus, while the conceptual approach of microcells remains the same, DSG-M must set more constraints to microcells so state can be propagated and modified properly.

In DSG-M, microcells are defined as rectangle shaped areas of pre-defined size. By using a Cartesian coordinate system, actors can be assigned microcell subscriptions that improves load balancing. Actors subscribe to microcells to show interest in read and/or writing to entities and the environment spaces of the subscriptions. Microcells are also used for interest management, to provide consistent update propagation. Subscriptions determine which actors should receive updates, based on the position where the update

occurred. If one actor creates, modifies, or deletes an entity, all actors subscribed to that microcell will receive the update. Actors not interested in updates for that microcell will be spared the update messages.

An example of a microcell assignment can be seen in Figure 1. A region of 256 meters squared, which is the standard region size of OpenSimulator and Second Life, is mapped into 8x8 microcells of 32x32 meters. We represent subscriptions as a colletion of Cartesian coordinates (X,Y). As an abbreviation, we also use $(X_0 - X_1, Y_0 - Y_1)$ to represent a collection of microcells that are contained in the larger rectangle of coordinates: $Rect((X_0,Y_0),(X_0,Y_1),(X_1,Y_0),(X_1,Y_1))$

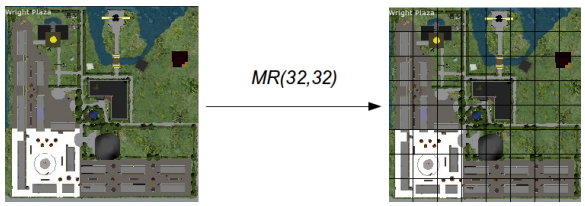

Figure 1: Two maps of the same 256 m^2 OpenSimulator region. The left figure shows an unmapped region. The right figure shows the region divided by microcells of size 32x32 m^2, mapping to Cartesian coordinates $(0-7,0-7)$.

The mapping and subscription sets of microcells in DSG-M may take any shape. It is common to have overlapping subscriptions of actors of different types. An example could be a script engine actor and a physics actor both subscribing to the same microcell. The script actor would handle script events, and the physics actor would process the physics Scene. Conflicts are resolved by a simple timestamp mechanism, enforced in DSG (**?**).

3.1 Active and Passive Subscriptions

To determine read or read/write permissions, each microcell subscription must be subscribed in one of two ways: active or passive. Active subscriptions to a microcell are subscriptions with read and write access to the state of the microcell, while passive subscriptions are read-only access.

Active microcell subscriptions give the simulator full control of the microcell's entities and environmental parameters (e.g. terrain), allowing any operations (i.e. creation, deletion, modification) to be performed and synced to actors subscribed to the same microcell.

Passive subscriptions are meant for read-only purpose: actors who are interested in updates but do not wish to operate on the microcell. Passive subscriptions are particularly useful if the actor needs to respond quickly to incoming entities from microcells outside of its subscription set.

A common example for passive subscription is in actors responsible for physics. Passive subscriptions are potentially useful in improving physics simulation correctness (e.g. collisions) near the borders of microcells, by allowing dead-reckoning of the entities in the passive microcells. The passively-subscribed physics actor will account for the entities of the passive areas, without broadcasting updates in that space to other actors.

3.2 State Propagation

State is propagated between actors in a subject-observer pattern, where each actor can be both a subject and an observer. At the start of the simulation, subjects inform observers of their microcell subscriptions. When an update is applied by an actor, it must notify all the relevant observers.

Notifications are triggered when an update is applied, due to one of two conditions: the subject actor created the update or the update came from another actor. Independent of how the notification is triggered, the actor determines the position in space where the update was applied. With the position, the actor determines the corresponding microcell coordinate. Finally, the actor attempts to match that microcell to

the microcell subscriptions of observers, obtained at the start of the simulation. The actor then sends the update to every observer subscribed to the microcell.

In our current design, actors are organized in a star topology. The actors are divided into two categories: **root actors** and **non-root actors**. Non-root actors can only send updates to root actors. Root actors have two responsibilities. First, they act as messaging hubs, receiving messages from non-root actors and forwarding them to observers. Second, they are persistent actors responsible for storing and maintaining Scene data. While other actors may be instatiated at runtime, root actors must always be available, to determine the set of valid microcells for updates, and for consistent state propagation. Root actors are also referred to as *persistence actors*. Further work on persistence actors can be seen in **?**.

An example of a star topology can be seen in Figure 2. Actor A is subscribed to the left half of the space, Actor C is subscribed to the right half, and Root Actor and actor B are subscribed to the entire space. The circle and triangles represent entities being updates.As an example, if an update is performed on the circle in Actor B, it is sent to the Root Actor, who forwards it to Actor A. If an update is performed on the triangle in Actor B, it is sent to the Root Actor, who forwards it to Actor C.

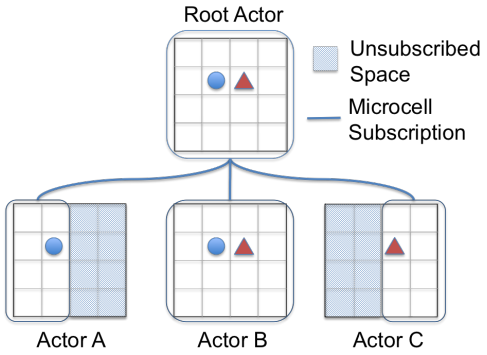

Figure 2: An example of a star topology with 3 non-root actors, and 1 root actor. The region is mapped as 4x4 microcells, the circle and triangles represent entities that were updated.

3.3 Crossings

If an actor is capable of generating position updates for an entity, it is possible that the entity will eventually cross the boundary of a microcell. The smaller the areas are partitioned, the more entity crossings are likely to happen between two partitions. Entity crossings can be costly operations: they may introduce the latency of packing, sending through a network, unpacking, and recreating the entity in the virtual world stack, and may increase the bandwidth due to shipping of a large amount of data across the network.

There are 3 possible situations where entity crossings may happen. Let m_1 be a microcell where the entity is originating from; m_2 be the microcell where the entity is heading towards; A_0 be actively subscribed to m_1 and is originating or forwarding the update that will cause the crossing; and $A_1...A_N$ be the list of actors connected to A_0, with subscription to either m_1 or m_2. For every actor Ai in $A_1..A_N$, the following situations are possible when a position update triggers a crossing at the origin A_0:

1. Ai **is subscribed to both** m_1 **and** m_2**.** This means that Ai is subscribed to the origin and destination microcells. In this situation, A_0 only forwards a simple update message with the new position, and locally updates the new microcell location for the entity. In Figure 3, this situation is illustrated by the crossing the circle entity in A_0. Actor A_i has both microcells 2 and 3, and thus receives only the update informing the new position and microcell.
2. Ai **is subscribed to** m_1**.** A_0 may send a simple microcell crossing update, since Ai has a copy of the full entity. Ai will forward the update to other interested actors, and will delete the entity. In

Figure 3, this situation can be seen by crossing the diamond entity in A_0. Actor A_i is subscribed to microcell 3, and was being updated on its state before the crossing. As in the previous situation, A_0 can send just the position and microcell update. Actor A_i will notice the entity is being crossed to a microcell outside of its subscription set, and will delete the diamond entity.

3. *Ai* **is subscribed to** m_2**.** *Ai* does not have a previous copy of the entity, since it was not subscribed to m_1. A_0 must encode enough information for *Ai* to build the entity locally. This situation is shown in Figure 3 by the triangle entity crossing, done by actor A_0. Actor A_i was not subscribed to microcell 1, so the crossing will generate a new entity. A_0 must send all the data necessary for A_i to rebuild the triangle entity in microcell 2.

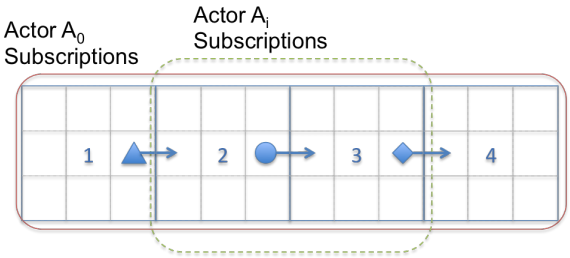

Figure 3: An example to demonstrate the 3 entity crossing situations. In the example, actor A_0 originates the crossing of the triangle, circle, and diamond entities, sending to actor A_i.

The above list pertains to the possible situations of the receiving actor. The sending actor A_0 is subscribed to $m1$ by definition, but not necessarily to $m2$. If A_0 is not subscribed to $m2$, A_0 sends the relevant updates to *Ai*, and deletes the entity locally.

4 EVALUATION

The first objective of our evaluation is to assess whether that DSG-M is capable of dividing and distributing the workload execution by space. We expect the distributed workload execution to yield similar results to the non-distributed execution. Additionally, we expect that overhead of inter-simulator communication is minimized so that the load of distributed simulators is close to the same load on a single machine, should such a powerful single machine exist.

In conducting the experiments, it became apparent that the simulation was affected by border effects, such as simulation collisions near the border and entity crossings. As such, the second objective is to test whether passive microcells reduce the border effects. Passive microcells should reduce the number of large entity crossings and provide updates of objects near the borders for improved collision detection, yielding improved results over simulations with active-only microcell subscriptions.

Our experiment consist of simulations of a real device, the *galton box* (**?**). A galton box is a board with pegs equally spaced in rows. The number of pegs per row increase from top to bottom by 1 per row, forming a triangular-shaped board of pegs. The simulated galton box can be seen in Figure 4.

The simulation workload is generated by 4 galton boxes set in the world, each 93 rows high, 128 meters wide. The bins are 1.2 meters wide each, and balls are of 1 meter diameter. At the top of each box there are 27 droppers, disposed in 3 rows of 9. Each dropper creates a ball right below it every t seconds, where t is the period.

Balls dropped at the top collide with pegs, and have a 50% chance to drop to the right or to the left peg in the lower row. The end result is, when the balls are collected at the bottom, the distribution of balls in bins will be a binomial distribution. With a large enough number of balls, the binomial distribution approximates a normal distribution.

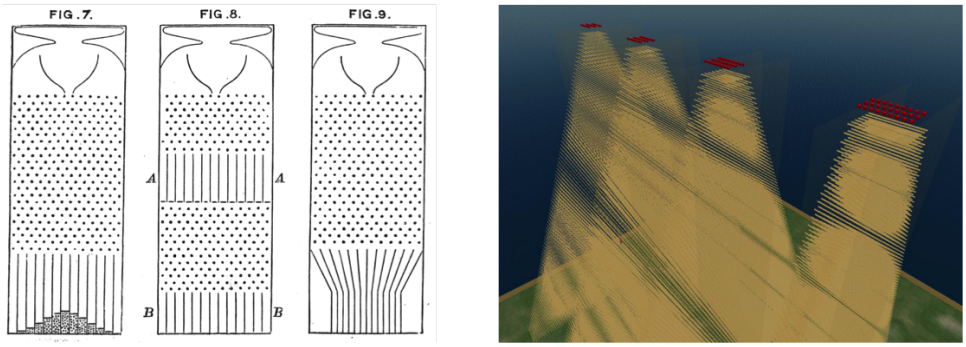

Figure 4: On the left, the original drawing for a galton box (**?**). On the right, the simulated galton boxes for the experiment. The boxes at the top drop the balls.

In terms of the evaluation, a normal distribution of balls is the measure of correct expected behavior. For purposes of evaluating DSG-M, we look into three metrics: (1) the error of the measured distribution of balls with respect to the expected distribution, (2) the average CPU load of the physics actors, and (3) the time it takes on average for balls to fall.

The first metric captures the precision of the simulation results. In pre-experiments, we identified two major sources of error, which we now control for: overwhelmed physics actor due to load, and inconsistency derived from space partitioning. The second and third metric captures the load placed on the simulator. By combining all three metrics, we can evaluate changes in the load of the simulators while controlling for the preciseness of the simulation's expected result. Furthermore, we identify the source of the error in precision, and test solutions to reduce it.

4.1 Experiment Description

The experiments run on a DSG-M implementation on top of the OpenSimulator platform, using a star topology. There are multiple actors on each experiment, of 4 different types:

- **Root actor**: The root actor, also known as persistence actor, does not perform any updates in the experiments. It is dedicated to coordinate actors, persist data, and forward messages. There is only one root actor in all experiments.
- **Physics actor**: The physics actor is responsible for creating physics updates, such as calculating positions and velocity based on forces and collisions. Depending on the experiment, there might be one or two physics actors present. The physics subscriptions varies per experiment.
- **Script actor**: The script actor is responsible for executing scripted actions. In the experiments, the script actor produces the balls at the top of the galton box on a regular interval period. In DSG only we use one script actor. In DSG-M, we use two script actors, covering two galton boxes each.
- **Client actor**: The client actor handles human observers. There is one client actor in every experiment, and one client connected to it. The client actor will not generate updates, but will receive every update from other actors.

All experiments use a microcell size of 4x4 meters. Each dropper releases 350 balls with a variable interval over the course of the experiment, for a total of 37,800 balls. Experiments were performed with dedicated desktops on a local area network. Each actor runs on a different desktop. The desktops are Intel Core i7-2600 CPU @ 3.40GHz, 16 GB RAM, and 1Gbps Ethernet controllers. Desktops run Ubuntu 12.10, and the application run with mono 3.2.8.

The experiments are divided into 2 parts. The first part of the experiments shows the effects of increasing load on results and physics beavior. The second part of the experiment measures the impact in performance and error of distributing load with microcell partitions on DSG.

4.2 Part 1: Non-partitioned Simulation

For this part, we use standard DSG, without microcell partitioning, and gradually increase the rate the droppers create balls to drop on the galton box. We test the resulting distribution and load measurements for periods of 12, 9, and 6 seconds. The topology for this part can be seen in Figure 5.

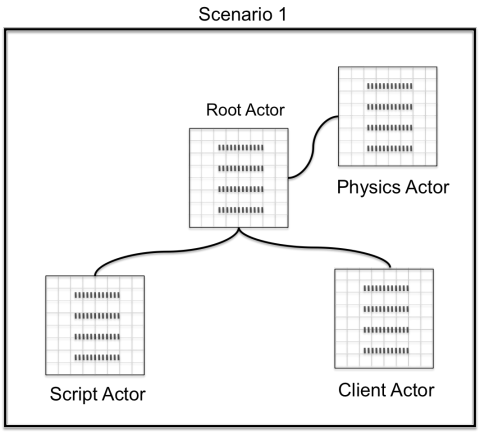

Figure 5: Topology and subscriptions of actors in the first part of the experiment, and in Scenario 1 of the second part. Each horizontal group of bars represent a galton box. Microcell grid not to scale.

4.2.1 Results

Table 1 shows the results for the first part of the experiment. The period is the time interval between each drop. The average load is measured as CPU percentage, where 100% CPU corresponds to one CPU core being fully used. Our machines had 8 cores, so the maximum possible CPU load is 800%. The fall duration is the time the ball takes between creation and colliding with the ground, and RMSE stands for Root-Mean-Square Error (RMSE), the square root of the mean squared error. RMSE uses the difference between the expected number of balls for a bin and the measured value in the simulation, for all 96 possible bins.

Table 1: Part 1: DSG without partitioning, load measurements. **AL**: Average load (CPU%), **PA**: Physics actor, **RA**: Root Actor, **FD**: Average fall duration of balls in seconds, **AB**: Average number of balls on the galton box, **RMSE**: Root-Mean error.

Period	AL:PA	AL:RA	FD(s)	AB	RMSE
12	244%	208%	125	1100	25.15
9	269%	204%	127	1479	26.25
6	317%	189%	295	4507	60.03

4.2.2 Discussion

This part of the experiment measures how physics load is handled by the simulator when the number of physical entities increases. Observing the results in Table 1, from a period of 12 seconds to 9, the CPU load increases, but the fall duration remains nearly the same. This indicates that the load was increased,

but the physics actor was capable of processing the updates in adequate time. However, from 9 seconds to 6, the fall duration increases by a factor of 2.32. The average number of balls also increases by a factor of 3.04. The CPU load increased only by a factor of 1.18. The increased fall duration and average number of balls are evidence that the physics actor is overwhelmed, and is dilating time. Dilation of time occurs when the simulation can no longer run at real-time speed, and the simulated world runs slower compared to real-world clock time. The average CPU increased to the limit of the simulator's parallel processing capability, and can no longer process the scene in time.

With the results of Table 1, we can also make a correlation between load and error. From 12 seconds to 9, the increase in the error is small, but still present. From 9 seconds to 6, the error is more than double. We also see that the error becomes much larger when the simulator no longer has hardware resources to manage the workload, and resorts to dilating time as result. Consequently, *we show that increased load is associated with increased error in the end distribution*.

4.3 Part 2: Space Partitioned Simulation

The second part of the experiment is to measure the improvement of microcells over DSG. We have determined that increasing the load on the physics actor to the point where it is overwhelmed causes the fall duration and error to increase significantly. We fix the dropper period at 6 balls per second, where the load exceeds one physics actor. We divided the second part into 3 scenarios:

1. **No space partitioning**: This is the base comparison scenario, extracted from part 1. Figure 5 represents the subscriptions in this scenario.
2. **Active subscriptions**: Two physics actors partition the Scene in half, splitting all 4 galton boxes. Figure 6 (left) shows the subscriptions of both actors in this scenario.
3. **Active and passive subscriptions**: Two physics actors partition the Scene, but each physics actor has **one** additional column of passive microcell subscriptions. Figure 6 (right) shows the subscriptions of both actors in this scenario.

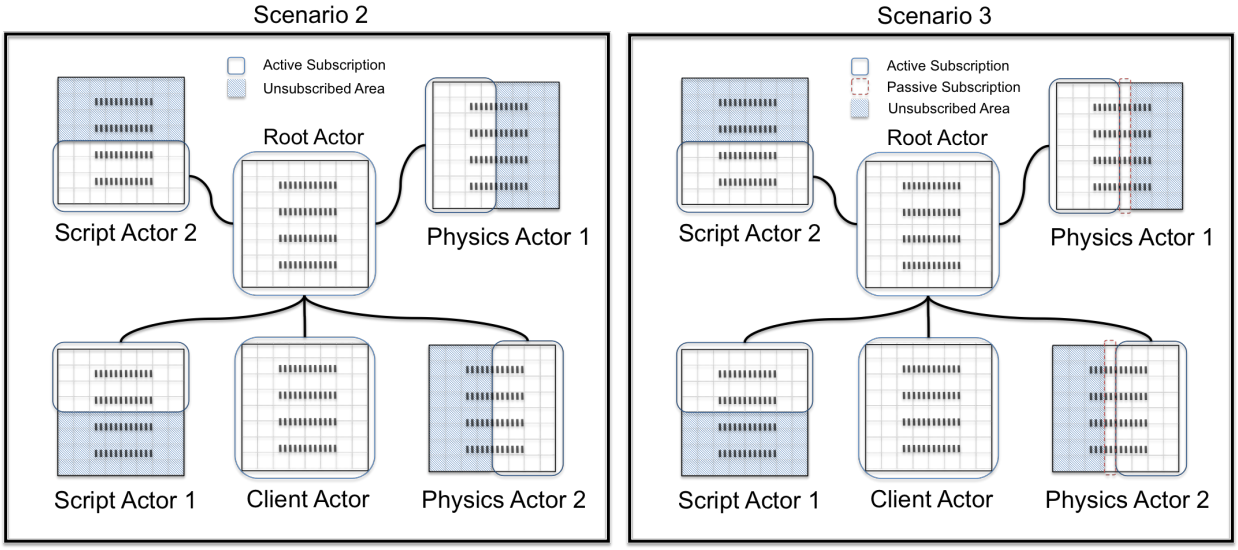

Figure 6: Topology and subscriptions of actors in Scenarios 2 and 3 of the second part of the experiment. Each horizontal group of bars represent a galton box. Microcell grid not to scale.

The first scenario serves as the base scenario for comparing the different scene partitioning approaches to load balancing. The second scenario determines the benefits, if any, of dividing the physics load in two

different processes partitioned by space. Finally, the third scenario adds passive microcell subscriptions at the border of the partition, which we hoped would improve the behavioral error.

4.3.1 Results

Table 2 shows the results of part 2 of the experiments. The average load and memory usage for each physics actor is shown. All the other measures are the same as in the first part.

We used no partitioning with a period of 12 seconds as the base scenario to generate the expected normal distribution. Figure 7 shows the resulting and expected distribution of balls for Scenarios 1, 2, and 3. The sum of all the measured and expected values of the normal curves match the number of balls in all three Figures.

Table 2: Part 2: DSG-M with space partitioning, at a period of 6 seconds per ball. **AL**: Average load (CPU%), **M**: Process memory in megabytes, **PA1 and PA2**: Physics actor 1 and 2, **RA**: Root actor, **FD**: Average fall duration of balls in seconds, **AB**: Sum of PA1 and PA2 average number of balls, **RMSE**: Root-Mean error.

Scenario	AL:PA1	AL:PA2	M:PA1(MB)	M:PA2(MB)	AL:RA	FD(s)	AB	RMSE
1	317%	—	1910	—	189%	295	4507	60.03
2	304%	246%	765	719	348%	265	1624	175.88
3	305%	279%	885	900	305%	126	3286	91.75

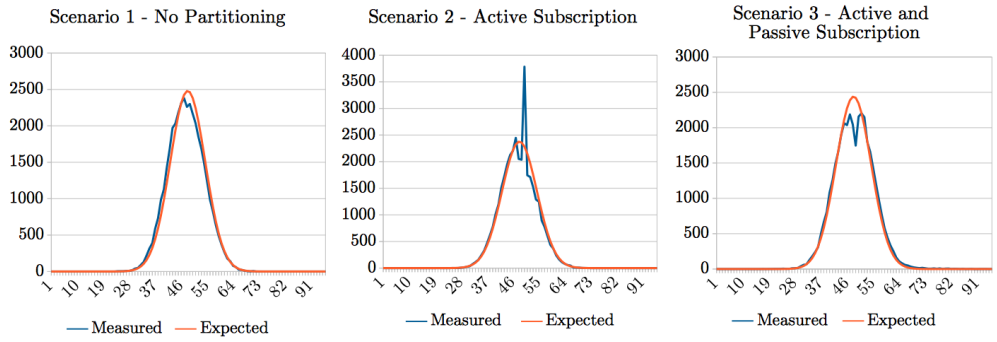

Figure 7: Expected and resulting distributions for Scenario 1 (no partitioning), Scenario 2 (active subscriptions only), and Scenario 3 (passive subscriptions included).

4.3.2 Discussion

In the second part of the experiment, we measure the effects of microcell partitioning on DSG. We have two goals: first, we wish to verify the improvement in scalability by checking that the load was divided with microcell partitioning. Second, we wish to assess whether the behavioral error of Scenario 2, where we use only active subscriptions, can be reduced by adding a passive subscription to microcells at the border.

We recall that the average load at a 6 second period for Scenario 1 was already overwhelmed, as indicated by a longer fall duration and a higher error, as seen in Table 1. Figure 7 shows the resulting distribution of the 3 scenarios of this experiment. In Table 2, the average loads for Scenario 2 and 3 are similar to Scenario 1, showing that in all scenarios, CPU resource is scarce. By looking at the fall duration however, we find evidence supporting our expectation for our first goal. Scenario 1 had a 295 seconds fall duration average. By partitioning the space into two, with active subscriptions, Scenario 2 had a decrease in the fall duration to 265 seconds. Scenario 3 decreased the fall duration even further to 126 seconds.

Both partitioned scenarios were successful in reducing the fall duration, and hence, in improving load balancing of the overwhelmed physics actor on Scenario 1.

In addition to a high CPU load, Scenario 1 allocated a considerable amount of memory, 1.9 gigabytes. In Scenarios 2 and 3, each physics actor used less than half of the memory of the single physics actor in Scenario 1. *The use of microcell partitioning in Scenarios 2 and 3 divided the memory usage over the single partition in Scenario 1.*

As for distribution errors, Table 2 shows Scenario 2 with a high RMSE. Objects near the border suffer imprecision when being crossed to another actor. By looking at distributions for Scenario 2, we see this imprecision as spike in the distribution near the border. By using passive microcells, that error is attenuated significantly, with a reduction in the RMSE to nearly half. For Scenario 3, we see a bimodal distribution closer to the expected curve. With a significant reduction in the error, *we show that passive microcells improved microcell partitioning by reducing the error in the expected distribution.*

5 CONCLUSION

We have looked into load balancing for virtual worlds, focusing on two dimensions: *Scene* and *operation* partitioning. In Scene partitioning, we partition the data and the state. In operation partitioning, we disaggregate the behaviors of virtual worlds.

In previous work on the DSG architecture, we successfully separated virtual world operations. In this paper, we design the *microcell* fine-grained Scene partitioning method to work on a distributed system with disaggregated operations, without loss of consistency.

In our evaluation, we demonstrate our load balancing technique through the use of active and passive subscriptions through a physics experiment. We successfully balanced an overwhelmed physics actor by partitioning the space in two, and attenuated the errors caused by the partitioning through the use of passive subscriptions. DSG-M proved successful in executing a workload that was infeasible with operation partitioning alone, while maintaining acceptable accuracy.

REFERENCES

Almroth, David 2010. "Pikko Server". Available at: http://www.erlang-factory.com/conference/ErlangUserConference2010/speakers/DavidAlmroth.

Aurenhammer, F. 1991. "Voronoi Diagrams - a Survey of a Fundamental Geometric Data Structure". *ACM Computing Surveys* 23 (3): 345–405.

Boukerche, A., and A. Roy. 2002, March. "Dynamic Grid-based Approach to Data Distribution Management". *J. Parallel Distrib. Comput.* 62 (3): 366–392.

Brandt, David 2005. "Scaling EVE Online, under the hood of the network layer". Available at: http://www.research.ibm.com/netgames2005/papers/brandt.pdf.

Buyukkaya, E., M. Abdallah, and R. Cavagna. 2009. "VoroGame: A Hybrid P2P Architecture for Massively Multiplayer Games". In *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, CCNC'09, 1100–1104. Piscataway, NJ, USA: IEEE Press.

Carlsson, C., and O. Hagsand. 1993. "DIVE A multi-user virtual reality system". *Proceedings of IEEE Virtual Reality Annual International Symposium - VRAIS '93*:394–400.

Dahmann, J. S., R. M. Fujimoto, and R. M. Weatherly. 1997. "The Department of Defense High Level Architecture". In *Proceedings of the 29th Conference on Winter Simulation*, WSC '97, 142–149. Washington, DC, USA: IEEE Computer Society.

De Vleeschauwer, B., B. Van Den Bossche, T. Verdickt, F. De Turck, B. Dhoedt, and P. Demeester. 2005. "Dynamic microcell assignment for massively multiplayer online gaming". In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games - NetGames '05*, 1–7. New York, New York, USA: ACM Press.

Galton, F. 1889. *Natural inheritance*, Volume 42. Macmillan.

Horn, D., E. Cheslack-Postava, B. F. Mistree, T. Azim, J. Terrace, M. J. Freedman, and P. Levis. 2010. "To infinity and not beyond: Scaling communication in virtual worlds with Meru". Technical report, Stanford University.

Kaplan, J., and N. Yankelovich. 2011. "Open Wonderland: An Extensible Virtual World Architecture". *IEEE Internet Computing* 15 (5): 38–45.

Lake, D., M. Bowman, and H. Liu. 2010. "Distributed scene graph to enable thousands of interacting users in a virtual environment". In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games - NetGames '10*, 1–6: IEEE Computer Society.

Liu, H., and M. Bowman. 2010. "Scale Virtual Worlds through Dynamic Load Balancing". *2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications*:43–52.

Liu, H., M. Bowman, R. Adams, J. Hurliman, and D. Lake. 2010. "Scaling virtual worlds: Simulation requirements and challenges". In *Proceedings of the 2010 Winter Simulation Conference*, WSC '10, 778–790. Baltimore, MD: IEEE Computer Society.

Lu, F., S. Parkin, and G. Morgan. 2006. "Load balancing for massively multiplayer online games". *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games - NetGames '06*:1.

Maxwell, D., H. Liu, R. Adams, and D. Lake. 2014. "Make Large-Scale Virtual Training a Reality". In *MODSIM 2014*, 1–11.

Stytz, M. R. 1996. "Distributed virtual environments". *IEEE Computer Graphics and Applications* 16 (3): 19–31.

Waters, R. C., D. B. Anderson, J. W. Barrus, D. C. Brogan, M. A. Casey, S. G. McKeown, T. Nitta, I. B. Sterns, and W. S. Yerazunis. 1997. "Diamond park and spline: A social virtual reality system with 3D animation, spoken interaction, and runtime modifiability". *Presence: Teleoperators and Virtual Environments* 6 (4): 461–480.

AUTHOR BIOGRAPHIES

ARTHUR VALADARES is a PhD Informatics student in the Bren School of Information and Computer Sciences at the University of California, Irvine. His research is centered in scaling virtual environments to accommodate thousands of users and virtual objects. He received his M.S. at University of California, Irvine, and his B.S. at Universidade Estadual de Campinas in Brazil. His e-mail address is avaladar@ics.uci.edu.

CRISTINA V. LOPES is a Professor of Informatics in the Bren School of Information and Computer Sciences at the University of California, Irvine. Her research focuses on software engineering for large-scale data and systems. She has a PhD from Northeastern University, and MS and BS degrees from Instituto Superior Tecnico in Portugal. Her email address is lopes@ics.uci.edu.

HUYAIU LIU is a research scientist in the Security and Privacy Research group, Intel Labs. She holds a Ph.D. in Computer Sciences from the University of Texas at Austin. While at Intel, she had worked on multi-radio networks, energy efficient communications, and scalable infrastructure for multi-user virtual environments. Her current research is in the domain of cloud security. Her email is huaiyu.liu@intel.com.