Empirical Evaluation of the Efficiency of Spatial Subdivision Schemes and Load Balancing Strategies for Networked Games

Peter Quax Jimmy Cleuren Wouter Vanmontfort Wim Lamotte
Hasselt University/tUL/IBBT
Wetenschapspark 2
3590 Diepenbeek, Belgium
Email: {peter.quax, wouter.vanmontfort, wim.lamotte}@uhasselt.be

***Abstract*—**
In this paper, we present an empirical evaluation of four spatial subdivision and three load balancing schemes that may be used for large-scale single-shard virtual environments, such as 3D networked Games and Virtual Communities. ALVIC-NG is an architecture that targets these environments and supports dynamic resource allocation. The scalability of an application based on ALVIC-NG and networked virtual environments in general is (among other factors) dependent on two metrics: the efficiency of the spatial subdivision scheme and the ability for the load balancing strategies to maintain a clustered assignment of regions to server instances. Both are discussed and evaluated in this paper through two modes of simulation. An offline version is used to decide on the most appropriate spatial subdivision scheme by creating a single-instance application that focuses on this specific metric. The online version implements the proposed strategies in ALVIC-NG and is used to both confirm the findings of the offline system and to determine the efficiency of the load balancing strategies by making use of several server instances and so-called bot servers to re-create the conditions of real-life deployments.

I. Introduction

Massive multiplayer online games (MMOGs) can roughly be categorized in two classes: those that use sharding (or instancing) and those that present players with a single virtual word. Prime examples of the first category include World of Warcraft and GuildWars. The second category is exemplified by Second Life and EVE Online. The choice for a specific approach is influenced by many factors, including technical ones (which will be discussed in this paper) and content-generation issues. Games that fall within the genre of MMORPGs are heavily dependent on the storylines and battle elements generated by game developers and do not allow user-generated content to be included in the game world - in order to guarantee the overall game experience and to avoid cheating. In these cases, a limited amount of levels, player characters and NPCs are introduced into the virtual world (because of the cost associated with generating this content). To support a large user community under these conditions, there is simply no other choice than to use a sharding or instancing approach.

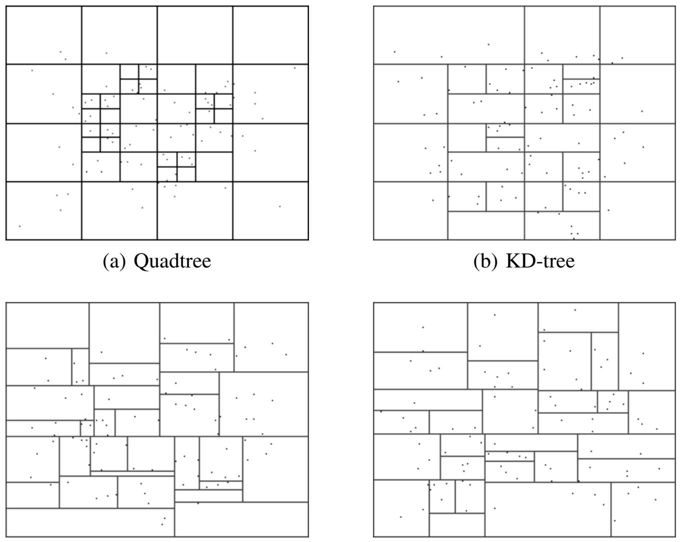
If user-generated content is deemed appropriate or even necessary for the business model or game genre (e.g. Virtual Communities), a single huge virtual world presents a number of benefits over the sharding approach. Interaction between participants is stimulated by allowing all players to co-exist. At the same time, the player experience is less limited, as the world to be explored 'feels' larger than in an instanced approach. Supporting such an architecture is however very challenging on a technical level, as each extension of the virtual world requires additional processing and network power. The same can also be said for sharding, however the main difference is in the fact that there is no communication between the servers that maintain the shards. To maintain a (seamless) single virtual world, there is need for cross-server migrations of players and content, creating a much harder challenge.

A typical approach for the single-world architectures is to assign geographical regions (in the virtual world) to specific server instances. The allocation is performed when levels are created by the game designers, and their size is determined by the expected player count and overall level design (e.g. corresponding to the dimensions of a room). An example of this approach is found in EVE Online with its Tranquility server cluster, where each star system is maintained by a single node.

Another approach to the problem stated before is to assign regions to servers based on a fixed spatial subdivision scheme (as in the case of Second Life). The problem with this way of working should be obvious: these solutions do not take into account either the extent or the number of players present in each section of the virtual world. Overcrowding and region abandonment make for both overloaded and underutilized servers in the cluster. From an economical point of view, this is clearly an undesirable situation.

In this paper, the focus will be on those architectures that allow for dynamic spatial subdivision schemes to be implemented. This makes for a better balance in the available processing power and bandwidth requirements versus the player distribution in the virtual world. More precisely, several of such schemes will be compared against one another in terms of their efficiency and overhead cost. This comparison is achieved through both simulations and empirical evaluations (using actual software instances in large numbers). As there are currently very few actual implementations of architectures that allow for dynamic allocation of server resources versus virtual world resources (at least those with freely available source code), the ALVIC-NG architecture (which is described in much more details in [2]) is used. By using the same overall system architecture for all schemes, they can be compared without varying factors besides the actual subdivision scheme. Also, the methods for load balancing between server instances are taken into account. Three proposed strategies are described and compared.

Dynamic resource allocation is a hot topic in networked virtual environment research, as it paves the way for deployment of server infrastructures 'in the cloud'. Using the cloud paradigm, additional server instances can be quickly provisioned on demand. By averaging out the load on these instances, a minimal number is required, which is interesting from an economic point of view (payment per instance). The interested reader is referred to [6] and [7].


(a) Quadtree (b) KD-tree

(c) Dynamic KD-tree (d) Limited KD-tree

Fig. 1: Proposed Subdivision Schemes

II. Tools and structures

A. ALVIC-NG

Although the ALVIC-NG architecture is described in detail in [2], its main features and properties required for the explanations below are summarized in this section.

Some of the basic building blocks of ALVIC-NG correspond to those found in nearly every client/server based architecture for large-scale single-shard games or virtual communities. The logic servers are responsible for maintaining an up-to-date state repository that describes the location of avatars in a specific region of the virtual world. At the same time, they perform calculations based on scripts that determine the behavior of NPCs and other objects present in the world. At any given time, a region in the virtual world can be under control of only a single logic server instance, while a single logic server instance may be responsible for maintaining the state of multiple regions. Moving an avatar between regions implies migrating its state between the server instances (which can be done in such a way that the process happens seamlessly in the eyes of the end-user). Other infrastructure such as database servers, asset servers and authentication services are not relevant for the discussion in this paper.

The unique feature of ALVIC-NG is the inclusion of proxy servers that perform a multitude of roles. First of all, they tunnel the message flow between the clients and the various logic server instances (present in the infrastructure for supporting the virtual world). That way, clients have a single point-of-contact for the entire virtual world, which aids in gaining access to the virtual world network infrastructure (e.g. NAT traversal, firewalls,...). Secondly, these proxy servers are responsible for detecting region boundary crossings for each connected client. If this happens, new connections are established if necessary and the information is retrieved from and sent to the relevant logic server instance. That way, the client can remain unaware of the actual spatial subdivision scheme used and is not impacted when the assignment of regions to servers changes. At the same time, the number of connections the logic server has to maintain is lowered, as only the proxy server instances make these connections (which can be re-used for clients present in the same regions). It has been shown in earlier work that the economic cost of introducing these additional server instances (the proxies) is outweighed by the advantages they offer. Their scalability is ensured as long as some properties are satisfied (see before and [2] for more details).

The Region Management System (RMS) has an overall view of the world, and keeps all proxy server instances up to date with the proper assignment of regions to logic server instances. It continuously probes logic server instances to find out what their load is; both in terms of system resources and higher-level information such as player count per region. If an overload or system malfunction is detected, the RMS triggers a split or merge process that redistributes specifically those affected regions over the server instances in the pool. It was explained before that such information is only disseminated towards the proxy servers and not to actual clients.

When a region split operation takes place, one might be tempted to distribute the newly formed regions over a set of different servers. In practice however, this is not a very good idea. The process of moving state and responsibilities between servers is resource and bandwidth intensive and results (unavoidably) in a short disruption of the player's experience. At the same time, the load on the server would drop significantly if the contents of an entire (large) region would be migrated at once, possibly followed by the relocation of smaller regions from other logic servers. This would quickly lead to extreme fragmentation of the virtual world, which is not a desirable condition. Therefore, when splitting a region, parts of it are kept local to the server that owned it originally, while responsibility for the remaining newly formed regions is handed over to other instances. The ways in which the areas are split are described in the next section.

B. Subdivision schemes

The schemes used for dividing the geographical space of the virtual world are related to several classic examples in use in Computer Graphics applications. Many more of them exist (e.g. based on Voronoi diagrams or Delaunay triangulation), but most are more suited for peer-to-peer approaches an require an update rate that is too high for practical purposes. Also, because it is up to the proxy servers to determine whether a client crosses a region boundary, it is essential that the cost associated with these calculations is kept low (in terms of CPU resources). Therefore, the choice was made to study those schemes that employ rectangular areas: *Quadtrees*, *KD-trees*, *Dynamic KD-trees* and *Limited KD-trees*. These will be briefly discussed in the next paragraph.

The *Quadtree* is a hierarchical subdivision scheme that divides regions into four distinct sub-regions as required. These subregions all have the same size and shape. The fact that this is a tree-based structure ensures that several subdivision levels may co-exist for the geographical space of the virtual world and facilitates the join operation for regions. An example of such a scheme is shown in figure 1a. In the *KD-tree* approach (see figure 1b), regions are split into two parts. The resulting regions are also of the same shape and size and the splitting process alternates between horizontal and vertical division. The above two examples do not take into account the meta-information associated with the regions, namely the number of clients present. In ALVIC-NG, the RMS has an overall overview of the load factors on the various server instances. Therefore, this information can be leveraged to design a more intelligent approach. Two of them are described here. Using *Dynamic KD-trees*, the region boundary is placed in such a way that approximately the same number of clients are present in each newly formed region. Regions are split into two new instances once the process is triggered. The main issue with this approach is that very narrow regions may be formed, which would lead to numerous region boundary crossings for players (which is stressful for the architecture and may be disruptive to the player experience). This is also illustrated in figure 1c. Therefore, in the *Limited KD-tree* solution (figure 1d), the dimension of the longest side of the newly formed region may only extend to x-times the length of the shortest side. For the simulations described below, this ratio is 3.

C. Load balancing

ALVIC-NG supports a pool of logic servers. It is the task of the RMS to ensure that the load is balanced over all of these instances, to prevent overloading or idle time on specific instances of the server infrastructure. For this purpose, a load balancing technique is required that takes decisions on which regions are to be placed on which logic server. The overall goals are to ensure that a cluster of adjoining regions is co-located on the same logic server instance, as the operation of moving state between server instances is heavy in terms of computational and processing overhead. At the same time, this will reduce the number of new connections that need to be established by the proxy server to various logic server instances when region boundaries are crossed. Three approaches are proposed for load balancing the regions over the logic servers: *round-robin*, *Cyberwalk* and *Lee*.

The most basic type of load balancing is to assign regions to logic servers in a *round-robin* fashion. Although this is very easy to implement and requires no information about the current state of assignments, it should be clear that this solution does not maintain the clustering of regions.

In case of the *Cyberwalk* load balancer, which is an adaptation of the one proposed in [1], the logic servers responsible for adjoining regions are queried for their current load. The server instance that is currently the least busy is used for transferring (part of) the regions onto, instead of simply choosing the next one in the pool. This load balancer exploits the run-time information of the servers and therefore is able to make better decisions (e.g. not to migrate a region to an already overloaded server or to ensure that the load is kept at an equal level on all instances). It also makes a better effort at maintaining the desired clustered region assignment to servers and limits fragmentation over many logic servers. Its main drawback however is its limitation to migrating a single region at a time. Also, care needs to be taken to ensure that regions do not 'flip' between logic server instances.

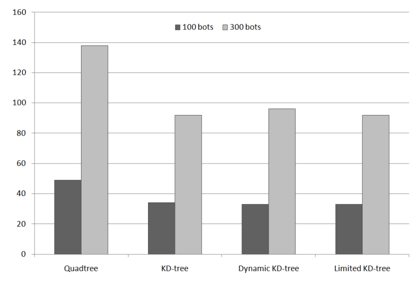
A third load balancer, based on the work of Lee & Lee [3] and which will be referred to as the *Lee* load balancer in this article, starts the same as the Cyberwalk balancer (i.e. by querying the neighbors). Once another instance is selected, an estimate is made on the load generated on both the existing and the new instance. In case this figure exceeds a threshold, a third server is asked to take charge of one or more regions. This sequence of actions is performed recursively until a distribution is found that satisfies the requirements, after which the actual migration (of several regions at once) is carried out. In case no distribution can be found, the process is postponed until a new logic server instance becomes available.

Several other load balancers are available, however the ones described above will be shown to be appropriate for use under the conditions presented in the sections on the simulation results. For some other options, the interested reader is referred to [4] and [5].

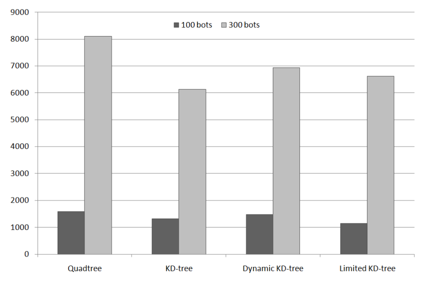
III. Offline simulations

A. Approach

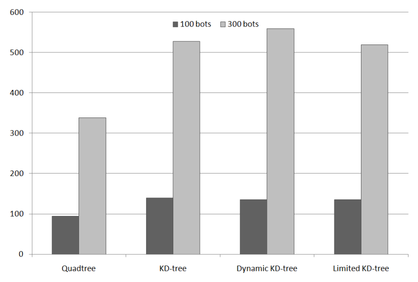
To evaluate the performance of the various schemes, an offline simulator was designed that emulates the behavior of human players in an extensive virtual world (so-called bots). These bots exhibit autonomous behavior of various sorts, which makes sure that they are dispersed throughout the world. First of all, the separation behavior from the Flocking model [9] is used to make sure that there are no collisions between them. Secondly, a slightly adapted version of the Random Waypoint Model [8] is used to assign a (temporary) movement



(a) Average number of regions



(b) Region crossings



(c) Scheme recalculations

Fig. 2: Results of offline simulations

goal to each of the bots by choosing destination waypoints located throughout the world. The adaptations are necessary to make sure that new objectives are chosen at random times, which eliminates unnatural lengthy movements along a straight line. The system ensures that more bots are located around the center of the virtual world versus the perimeter, which also accords to real-life situations. Although more lifelike movement patterns do exist, the chosen approach allows for fast calculations and results in the required number of region border crossings. The fact that the exact movement pattern does not influence the outcome will be supported through the on-line simulations.

In these offline simulations, the decision on when to split or merge regions can be done by the simulation software itself (it would be under control of the RMS in case of ALVIC-NG). A hysteresis is built-in to ensure that regions are not split or merged based on the boundary crossing by a single bot.

An important remark needs to be made regarding the region join/merge operations that need to take place whenever the occupation of that region drops below the threshold. One might be tempted to create new regions that do not align with the original subdivision scheme. Under normal conditions (for the tree-based structures) children of a specific node are merged into the parent node. This ensures that the world continues to consist of rectangular regions, while the arbitrary join/merge operations would quickly result in non-regular regions. The reader is reminded that the proxy servers are charged with performing boundary crossing calculations for all connected clients. It is therefore essential that this is done in a very efficient manner, to ensure scalability. Retaining a rectangular grid pattern facilitates calculations and results in a less fragmented subdivision scheme, the importance of which will become clear in the explanations on the online simulations in section IV.

Overall, simulations were run five consecutive times and the results were averaged to obtain the figures in this section. Some statistics on the conditions of the tests: each bot is able to take 2000 steps and chooses between 20 random waypoints; each region is able to support 5 bots and has a hysteresis of 1. The simulations were performed on two population sizes: one of 100 bots and one using 300 instances.

As the results of the load balancing strategies are only really useful for the online simulations, they are not implemented in the offline scenario.

B. Results

In figure 2a, the average number of regions created is represented for the four subdivision schemes. It is immediately apparent that the Quadtree implementation suffers from the fact that it immediately decides to split every node into 4 children and takes decisions independent of the context (i.c. the location of the avatars within the regions). It was indicated before that a high fragmentation ratio of the regions was to be avoided, because of the difficulties associated with the assignment of these regions to servers in a clustered fashion. The three other approaches exhibit a better behavior, with a slight disadvantage for the Dynamic KD-tree approach (due to the absence of a region size limit and subsequent creation of very narrow regions).

The latter has an important impact on the number of region boundary crossings, shown in figure 2b. As these are resource-intensive operations, they are to be avoided when possible. It is also clear from the figure that the Quadtree scheme results in an even larger number of crossings (directly related to the number of regions). A possibly surprising result is that the standard KD-tree proves to be more efficient when compared to the Limited KD-tree (which does take into account the player load). Apparently, deciding on where to create new boundaries at the time of the split operation is not representative for the further dispersion of the bots throughout the newly created regions.

In terms of the number of split/merge operations carried out (see figure 2c), the Quadtree proves to be the best choice, which is an unsurprising result (as it also creates the largest number of regions). There is a 2:1 ratio compared to the KD-tree schemes in terms of directly created child nodes and conversely in the capability of the regions to support a number of avatars. There is relatively little difference between the latter approaches, although the Dynamic KD-tree again proves to be the worst choice.

Overall, there is a clear preference that can be derived from these experiments, namely to choose the standard KD-tree as basis for the subdivision scheme. The additional overhead associated with the Limited KD-tree (keeping track of the location of the avatars at time of the split operation) is not converted into a significant benefit with regards to the three factors described above. In section IV-B, these results will be compared to the ones obtained from the online simulations.

IV. Online simulations

A. Approach

To see whether the results from the offline simulations matched those under more realistic conditions, an online version was implemented that actually runs on the ALVIC-NG architecture. For a detailed description of the bot setup, the reader is referred to [2]. The overall approach is similar to the one used in VENUS[10] but targeted towards the specific structure of ALVIC-NG. In summary, a cluster of machines is used to run instances of the client software without visualization. These bot servers instruct the avatars to move in a pre-defined pattern, to generate realistic position updates and to generate network traffic that is sent through the infrastructure.

For the purposes of this setup, the avatars were instructed to walk around in circles of varying radius. This is a deviation from the movement patterns in the offline scenario, but it also provides a good test case to ascertain that the exact movement pattern does not influence the outcome of the experiment. Although there is some loss of control over the various instances of bot servers (when compared to the offline simulation), the overall trends that emerge are supposed to remain the same. The simulations detailed here are performed using 1500 simultaneous bots. Each regions supports 20 simultaneous clients. Although ALVIC-NG supports much larger amount of users than this, for purposes of this paper the virtual world size was adapted to these figures.

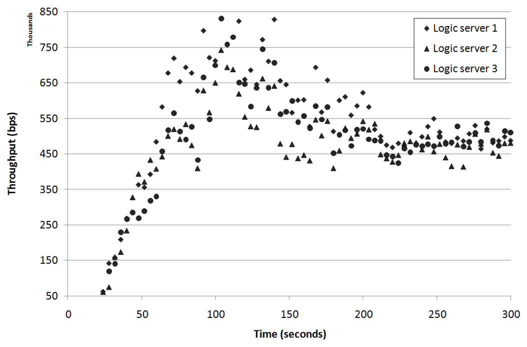
B. Spatial subdivision schemes

Overall, it can be observed from figure 3 that results are similar to the offline simulations. Again, the Quadtree scheme proves the least optimal choice. For KD-trees, splitting the regions depending on the distribution of the avatars at the time of the split proves to be only slightly beneficial (in terms of the number of regions formed (figure 3a) and the number of transitions between regions (figure 3b). However, there is additional overhead associated with such a subdivision scheme and there is an increase (as can be seen in figure 3c) in the number of recalculations required. These factors lead to the conclusion that the balance tips in favor of the traditional KD-tree approach, confirming the results obtained from offline simulations.

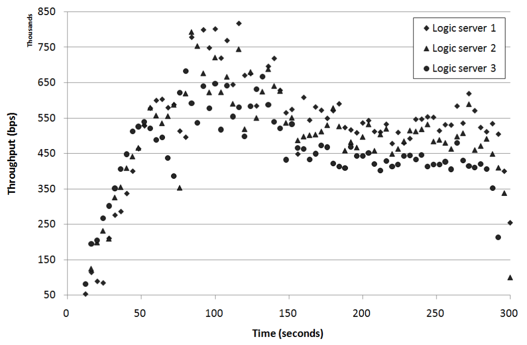
C. Load balancing strategies

To ascertain the performance of the load balancing strategies, two metrics are relevant. First. the throughput of the individual server instances is important. In a load-balanced environment, these figures should be comparable between instances with the exception of short bursts (which would be the triggers for region splitting). Secondly, the clustering of regions is important, as this will limit the state transfers between logic servers when region boundaries are crossed. The conditions under which the test results were obtained are as follows: the simulations were run using 3000 active bots. Three logic servers were used, 6 proxy servers and 6 servers for generating the virtual avatars. Bots are spawned one after the other until the total amount required is reached. The Limited KD-tree is used as spatial subdivision scheme.

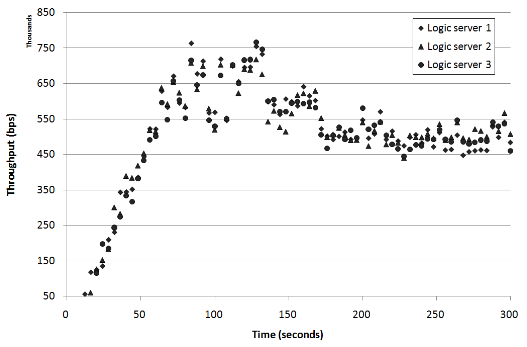
From figure 4a, it is apparent that the round-robin load balancer does not do particularly well in maintaining an equal load over the logic servers. Regions are re-assigned to other instances regardless of their current load. Also, there is no estimation of the drop in load on the current server, which would be useful to determine the number of regions to be transferred (instead of maintaining them on the currently active



(a) Round-robin

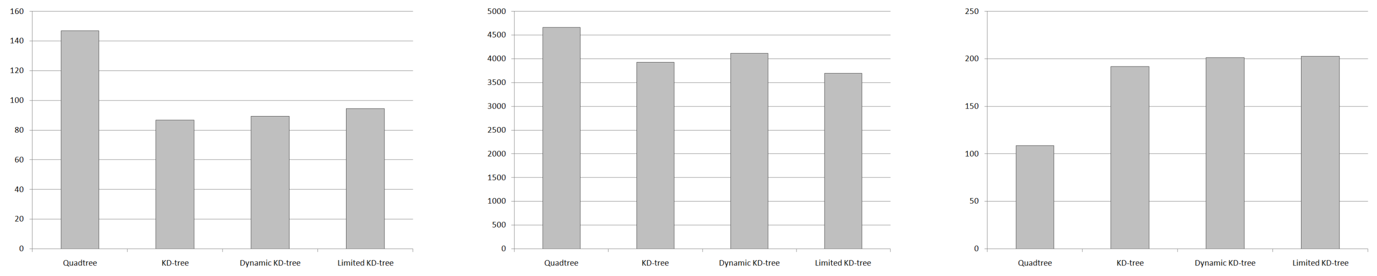


(b) Cyberwalk



(c) Lee

Fig. 4: Logic server throughput comparison under different load balancing strategies



(a) Average number of regions (b) Region crossings (c) Scheme recalculations

Fig. 3: Results of online simulations

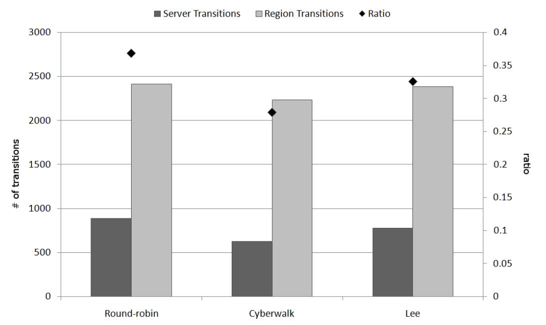


Fig. 5: Region changes under different load balancing strategies

instance). The Cyberwalk load balancer performs marginally better (figure 4b). It still lacks the ability to transfer the split regions to more than one server instance and is only able to migrate to one server at a time, which results in a delayed convergence to the ‘optimal’ distribution. Finally, the Lee load balancing strategy is shown in figure 4c. Obviously, this is the optimal choice for the purpose under this metric, as it ensures an equal distribution of the load over the three instances.

It was indicated above that clustering of regions to logic servers is a second important metric. This is measured using the number of bots that have to switch between logic server instances during the execution of the simulation. The results are shown in figure 5. Indicated are the total number of region transitions and the number of changes that actually involve a transfer of state between logic server instances (server transition), along with the ratio of both values. Although the Cyberwalk strategy seems to perform better, this can be explained by the fact that this strategy is slower at converging to the ‘optimal’ condition when compared to the Lee load balancer. However, the state of the virtual world is continuously changing, which means in practice that the ‘optimal distribution’ is also ever-changing. A slower converging method would be more able to deal with those conditions than one that tries to converge immediately. It is hard to pick a winner between these two alternatives; the optimal choice depends on the type of game that is supported by the architecture (fast-paced action versus slow-paced exploration).

V. Conclusion

In this paper, we have presented an empirical evaluation of the efficiency of three spatial subdivision schemes and load balancers that may be used for single-shard virtual environments through offline and online simulations. The initial findings from the offline simulations were that the KD-tree proved a better choice than the Quadtree as a subdivision method. The additional efficiency gained by the more elaborate subdivision schemes were proven not to be substantial enough to offset the overhead associated with the required calculations. The results obtained from these experiments were confirmed in a second setup, in which the implementation of ALVIC-NG was used to simulate an environment with thousands of users. Also, the Cyberwalk and Lee load balancing strategies (which determine what regions are placed on a specific server instance) were indicated to be useful candidates, the choice between them depending on the game genre.

References

[1] A. Steed and R. Abou-Haidar, “Partitioning crowded virtual environments,” in *Proc. of VRST ’03*. New York, NY, USA: ACM, pp. 7–14.

[2] P. Quax, J. Dierckx, B. Cornelissen, and W. Lamotte, “Alvic versus the internet: Redesigning a networked virtual environment architecture,” *International Journal of Computer Games Technology*, vol. 2008, no. 594313, 2008.

[3] K. Lee and D. Lee, “A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution,” in *Proc. of VRST ’03*. New York, NY, USA: ACM, pp. 160–168.

[4] J. Chen, B. Wu, M. Delap, B. Knutsson, H. Lu, and C. Amza, “Locality aware dynamic load management for massively multiplayer games,” in *Proc. of PPoPP ’05*. New York, NY, USA: ACM, 2005, pp. 289–300.

[5] 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 *Proc. of NetGames ’05*. New York, NY, USA: ACM, pp. 1–7.

[6] M. T. Najaran and C. Krasic, “Scaling online games with adaptive interest management in the cloud,” in *Proceedings of NetGames ’10*. Piscataway, NJ, USA: IEEE Press, pp. 9:1–9:6.

[7] M. Sung, “Keynote speech at netgames 2010 : From online gaming to cloud computing,” 2010.

[8] D. B. Johnson and D. A. Maltz, “Dynamic source routing in ad hoc wireless networks,” in *Mobile Computing*. Kluwer Academic Publishers, 1996, pp. 153–181.

[9] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *SIGGRAPH Comput. Graph.*, vol. 21, pp. 25–34, August 1987.

[10] Y. Jung, B.-H. Lim, K.-H. Sim, H. Lee, I. Park, J. Chung, and J. Lee, “Venus: The online game simulator using massively virtual clients,” in *LNCS : Systems Modeling and Simulation: Theory and Applications*. Springer Berlin / Heidelberg, 2005, vol. 3398, pp. 589–596.