Load Balancing for Virtual Worlds by Splitting and Merging Spatial Regions

Umar Farooq
University of Science and Technology Bannu, Pakistan
E-mail: umar@ustb.edu.pk

John Glauert
University of East Anglia Norwich, UK
E-mail: j.glauert@uea.ac.uk

Kashif Zia
Sohar University, Oman
E-mail: kzia@soharuni.edu.om

Keywords: virtual world, OpenSimulator, spatial partitioning, under utilisation, over utilisation

Received: July 7, 2017

The aggregation algorithm, an integral part of our dynamic infrastructure (using an expansion and a contraction model) for managing scalable virtual worlds, was proposed in our previous work, to overcome the limitations of the current methods using static and hierarchical approaches. The basic aim was to get two contiguous spaces made of smaller regions while distributing the load as balanced as possible among two servers. This algorithm performs well for the perfect square shaped spaces but fails when it is applied to spaces of other shapes. The current merging algorithms also assign non-contiguous spaces to servers during the contraction phase. This is due to the unavailability of an explicit continuity check in both aggregation and merging algorithms.

In this paper, we provide state-of-the-art in scaling virtual worlds and outline their limitations. It provides both theoretical arguments and simulation results that contiguous spaces have potential benefits. This work, then, extends both the aggregation and merging algorithms and incorporates an explicit continuity check to cope with the issues introduced by allowing non-contiguous spaces. It is demonstrated with the help of results from our prototype that the extended methods strictly achieves the theoretical goals of the proposed methods.

Povzetek: Podan je pregled skalirnih metod v navideznih svetovih (VWs) in nov algoritem za razširjanje in krčenje podprostorov.

1 Introduction

Virtual Worlds (VWs) are the most advanced Virtual Environments (VEs) that allow users to immerse into 3D shared spaces. They provide real or imaginary content and users in them are represented by digital characters called avatars [14]. VWs are interactive and collaborative environments that have distinguishing features such as coherence and persistence. They are general purpose and social in nature [21, 24]. They have attracted huge attention of individuals, businesses, and organisations of various domains such as entertainment, design, government, and research and development communities. They are becoming a major tool for collaborative activities [16, 1]. Second Life (SL) [24, 27] is state-of-the-art in commercial VW development frameworks and it imitates the physical world. It is extensively used for content development by various communities such as business and entertainment industries. The research and development community has, however, shown more interest in OpenSimulator (OSm) [13, 20] - an open source alternative to SL.

Scalability is the major issue to dealt with in VEs. Traditionally, it is achieved by splitting the whole virtual space and assigning it to a set of dedicated servers for simulating it [19]. Game environments are easily scalable as they exploit the concept of sharding that allows the duplication of content [21]. However, the space in VWs, is distributed using spatial partitioning. VWs do not allow duplication of content as they have to maintain a unified coherent space [19, 21]. VWs are very complex as they integrate in them the challenges of many hard simulation problems such as large scale, real time computation and communication using a simulation centric architecture developed for standard simulation environments [21]. Therefore, they are much restricted and are able to host only a limited number of players per Simulator (Sim) [15].

Static and dynamic methods are currently in practice to assign a virtual space to a given set of servers. While a system is up and running, a statically assigned space never changes and manual reconfigurations are required to incorporate changes in current allocation. On the other hand, dynamic spatial partitioning allows re-assignments while

the system is running. This process, however, is too expensive as it involves transferring both content and players. Dynamic techniques are usually categorised into flat and hierarchical mechanisms. Flat mechanisms use either a local, global, or an adaptive strategy for load distribution [4]. Hierarchical approaches adopt a parent child hierarchy for managing resources. In our previous work, we developed a hybrid infrastructure comprises an expansion and a contraction model to cope with the issues in both static and dynamic mechanisms presented in section 2.1.1 and 2.1.2 [6, 8]. It proposed an aggregation and assignment algorithm [7, 9] for the expansion phase and merging algorithms for the contraction phase [6]. The major goal of both types of algorithms was to provide contiguous spaces for a Sim to host.

In this paper, we present the critical analysis of some of the well-known static and dynamic methods currently in use for scalable VEs including our proposed framework. It provides justification for using the continuity in spaces assigned to different Sims. It determines the limitations in both aggregation and merging algorithms and, then, extend them to overcome these limitations. Simple illustrations are used to show that the extended models successfully assign contiguous spaces and avoid non contiguous spaces.

The rest of the paper is structured as follows. Section 2 provides the Literature review, background and motivation for this work. The justification for using the continuity constraint in expansion and contraction phases is provided in section 3. The basic and extended versions of the expansion and contraction algorithms and their illustrations with examples from our prototype are presented in section 4 and 5. Finally, section 6 concludes the paper and provides future directions.

2 Background and motivation

2.1 The Literature

The mechanisms for scaling VEs found in the Literature can be categorised as static, dynamic, and hybrid in nature. These mechanisms are critically analysed in this section.

2.1.1 Static mechanisms

The underlying infrastructures for SL and OSm called SL Grid (SLG) [24], and OSm Grid (OSmG) [20] extend the Butterfly Grid (BG) [17]. They use static assignment for an improved performance and avoid the expensive transferring activities. SL architecture is much restricted and it allows a server (usually, a Simulator (Sim)) to host only up-to a maximum of four regions. OSm uses the extended architecture of SL proposed by the Linden Lab [24] and is, therefore, more open than SL. It allows a Sim to manage an arbitrary number of regions but the environment remains static. SL and OSm both lack dynamic adjustments and, therefore, introduce resource provisioning issues. Resources in this arrangement are greatly misused. Resources in some cases, when no players are visiting the content assign to them, might remain under-utilised - this case is termed as over-provisioning. On the other hand, system capacity is restricted as no additional resources are available when more players are interested to join a space - this is termed as under-provisioning [29].

2.1.2 Dynamic mechanisms

To cope with the issues in static assignment methods, a number of dynamic strategies are developed that are broadly categorised as flat and hierarchical in nature. Load balancing in mechanisms using flat orientation uses either a local, global or an adaptive strategy. Local strategies (such as the one adopted in [25]) are not scalable as each server is capable of sharing its load only with the neighbouring servers. They fail to scale when neighbouring servers are also overloaded. Global strategies (such as those used in [23, 28]) use complex procedures to re-distribute the workload evenly on all the servers and thus degrade interactive user experience. They are not suitable for those systems that involve frequent re-adjustments. Adaptive strategies adopt the simplicity of the local but the scalability of the global strategies. They scale better than local strategy as a server extends sharing its load with the servers next to the neighbouring servers, in case the neighbouring servers are also overloaded. Further, they are less complex than global strategies [22].

VEs prefer using hierarchical approaches which are, generally more flexible and scalable than flat mechanisms, as flat mechanisms put extra burden of user migration on the system [26]. Hierarchical methods (such as those presented in [18, 3, 2, 5]), however, suffer from complexity, latencies, and poor performance as they places no restrictions of the size of content assigned to a server and the levels in a resource management tree [6].

2.1.3 Hybrid mechanism: state-of-the-art in scalable VWs

In our previous work, we presented a dynamic scalable infrastructure and introduced the concept of a hybrid grid infrastructure for its implementation. When the load is normal, this hybrid mechanism behaves like a static grid infrastructure in which each Sim is hosting its assigned space. As the load increases, it dynamically adds additional resources at lower levels to cope with increasing load. The basic aim was to overcome the limitations of existing static and dynamic mechanisms. The proposed mechanism achieves this using an expansion and a contraction model. The expansion phase includes the split, aggregation, and assignment methods. The contraction phase provides two variation for merging process.

In this work, each server in start, handles almost a square shaped space and a regular square pattern is used to split the overloaded space. The number of players a server can potentially host is represented by SimCapacity. However, it

initiates a split operation based on a parameter called SplitCapacity. MergeCapacity parameter is used by a server to initiate a merge operation [8].

The Expansion Phase (Splitting)

The Split Process: When a Sim gets overloaded, it divides its assigned space into an equal sized sub-regions (normally either 4, 9, or 16 onwards) that achieves regions whose density is less than the SplitCapacity and thus eases the load but against a boundary condition. A region representing an un-partitioned but varied size of space is divided during a split operation if it is not the ultimate space that cannot be further partitioned.

The Aggregation Process: uses an aggregation algorithm [9] to determine two aggregates of the smaller spaces comprising an assigned space, provided as input by the Split Process. It aims to minimise resource utilisation, and communication and implementation cost. It tries to obtain aggregates with fair load by combining adjacent regions and avoiding the diagonal ones. It combines only those regions (even those in a diagonal) sharing physical boundaries with the regions already in an aggregate. The main objective is to obtain two contiguous areas for assignment to minimise the number of connections/disconnections between servers when players move between regions. The levels in the management tree are minimised by placing all servers handling regions obtained in a split as siblings.

The aggregation algorithm takes input in the form of a tiled grid. Keeping its goals in mind, it takes any two consecutive corner regions to start aggregation with. It uses four aggregation strategies, namely, **Row by Row (RR)**, **Column by Column (CC)**, **Row and Column in Turn (RCnT)**, and **Row and Column in Turn with Diagonal (RCnTwD)** which guarantee examining the entire set of unique and valuable combinations.

The Assignment Process: assigns one of the aggregates determined in aggregation step to an additional server. The current implementation transfers the aggregate with less regions and smaller number of players. Each server that is hosting an aggregate maintains the identity of smaller regions which are, then, re-assigned at later stages based on an increase in load until each of them is handled by an individual server. The split process is repeated at this stage on smaller regions unless the boundary conditions are met.

The Contraction Phase

The contraction phase implements the merging process and it ensures that the resources are utilised as per the requirements. Merging is triggered by a server when it notices a decrease in the number of players it manages. In current implementation of our work, a Sim is either a parent or a child. However, only a child Sim initiates a merge process.

Contraction allows two merging strategies called, Parent Merge (PntMrg) strategy and Child Merge (ChMrg) strategy. In PntMrg strategy, a child Sim initiates a merge operation only if it can return its full load to the parent Sim. However, it is believed that the system potentially holds the resources for more time and it is not efficient in terms of resources. This issue is resolved in the ChMrg strategy. In ChMrg strategy, a child Sim relocates its full load to one of its siblings, if it is unable to integrate the load with the parent Sim. When a Sim capacity goes beyond MergeCapacity, then it checks for an appropriate Sim (the parent or a sibling based on the strategy being used) and the merging is initiated if and only if, the cumulative load of both the Sims is less than or equal to the MergeCapacity. In case of a successful merge, the Sim who initiated the merge releases itself.

The Implementation, Worth and Limitations of Hybrid Grid Infrastructure

Non existence of a specialised framework for developing highly scalable VWs motivated us to develop the hybrid grid infrastructure. The main goals were to assign coherent contiguous spaces using a resource management tree with minimum additional levels for an improved communication and implementation cost while distributing the load as balanced as possible. We used OSm framework for the implementation of this work. Since, the basic architecture does not support dynamic capabilities, we extended the OSm architecture to support dynamic scalability [10]. We investigated the basic capabilities for various activities involved in the expansion and contraction phases and extended some of the costly activities [11]. We, then, developed a working prototype of this infrastructure using OSm framework by utilising its basic and extended methods [6, 12]. It moves the players in a transferring region into a transit region during the re-allocation process.

Our hybrid infrastructure achieved improvements against both static and dynamic mechanisms in multiple dimensions described using a set of parameters including scale, resource utilisation, complexity, communication and implementation costs, and interactive user experience. When compared with static assignment method that assigns multiple regions to a Sim, the proposed method scale beyond the capacity of static assignment. However, it scales exactly up-to the same capacity as the static method in which each Sim hosts a single region. In both cases, resource utilisation is improved by starting a Sim with more regions and assigning additional resources purely on current workload. The proposed mechanism, therefore, solves over-provision and under-provision of resources. By adopting a localised decentralised approach and reducing the levels in the resource management tree, it greatly reduces complexity, and communication issues. Since, the players never go off, it improves their interactive user experience. Various concepts of OSm framework and the extended methods developed for various activities involved in re-allocation process greatly reduced the implementation and transferring costs.

During the implementation of our work, we discovered that the aggregation algorithm determines the two contiguous spaces when it is applied on a square shaped space. However, it fails to get contiguous spaces when it is applied to spaces of other shapes. Similarly, the merging algorithms also permit a merge of non-contiguous spaces, a clear violation of the basic goals set earlier for our scalable

infrastructure.

2.2 Motivation, goals and contribution

Hybrid grid infrastructure got improvements in multiple aspects discussed above, however, the limitations in its current implementation greatly restricts its functionality. To get hold of the benefits of the proposed infrastructure motivated us to extend its aggregation and merging algorithms. The main goal of this work is to enable these algorithms to produce contiguous spaces for any shape of spaces comprises various regions. It also aims to justify the use of continuity in assigning spaces to servers.

This work reports justification for the contiguous spaces, and the extended algorithms for aggregation and merging followed by their illustrations with results from our implementation.

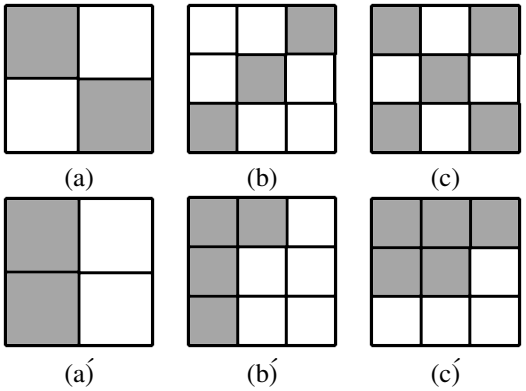

Figure 1: Odd and their equivalent valid aggregates. (a) Aggregates based on diagonals for a 4-region world; (á) Valid aggregates for Figure(a); (b) Aggregates based on a single diagonal for a 9-region world; (b́) Valid combinations for Figure(b); (c) Aggregates based on both diagonals for a 9-region world; (ć) Valid combinations for Figure(c).

3 Evaluating continuity model

This section provides justification and the benefits of the continuity model to be incorporated in basic aggregation and merging algorithms. It shows how odd and isolated cases introduce extra burden in terms of communication, implementation, and user migrations. Three parameters are used for this evaluation that are: total number of regional boundaries exposed to the external regions; total number of isolated regions managed by a single Sim; and number of user crossings between different Sims.

Three example odd cases (from a wide range of possible combinations) which are presented in Figure 1(a)-(c) are used for evaluation and comparison with equivalent valid aggregates presented in Figure 1(á)-(ć). The regions in one aggregate in Figure 1 are marked black and white in the second aggregate.

3.1 Theoretical evaluation

Current VWs treat each region as a complete isolated system and, therefore, introduce complex boundary crossings between the regions regardless of the fact that they might be on a single Sim. The concept of mega-regions is introduced in OSm to get bigger spaces and reduce intra-sim communication. It also help in reducing the number of crossings between the regions. However, the current mega-regions only integrate the neighbouring and contiguous regions. It is difficult to take advantage of this exciting feature of OSm framework when isolated regions are allowed. The inclusion of continuity model in aggregation algorithm thus allow us to get benefit of mega-regions during implementation.

Two parameters that are: the number of isolated spaces managed by a Sim, and the number of boundaries in an aggregate exposed to regions of other aggregate are used to provide theoretical justifications. Table 1 provides results for these parameters where it can be seen that non-contiguous spaces normally provide a large number of isolated spaces. However, the inclusion of continuity model reduced them to only and only two contiguous spaces. Excluded cases greatly increase the implementation complexity by managing different isolated areas compared with valid combinations. Similarly, communication and interaction in valid combinations are significantly reduced compared with odd cases. It can also be noted that when a system has more isolated regions, it generally increases the number of regional boundaries exposed to players of the external regions. It implies that the players have more spaces and chances to go across a Sim boundary to another Sim served by a different server. It potentially increases communication among regions on the same Sim. The next section justifies this claim using a simple simulation environment in terms of players crossing the boundaries between different Sims. Overall, about 50% decrease is achieved in terms of number of exposed boundaries by selecting valid combinations by the extended algorithm as shown in Table 1.

3.2 Simulation based evaluation

The most common parameter in scaling a parallel and distributed system such as a VW is to determine, how much the distribution process increases the number of crossings between the servers in a given system.

3.2.1 Simulation environment

The console window is partitioned into regions based on aggregates and different colours are used to represent the valid and odd combinations as shown in Figure 1. In each case, the odd and its corresponding valid combination are simulated for the same duration against the capacities including one, five, and ten randomly distributed objects (representing players). Each object is allowed to select a random move in one of the four directions at each step

Table 1: Comparison of isolated spaces and their exposed boundaries for both odd and valid aggregates.

Case	Description	Number of isolated spaces	Number of Exposed boundaries
1	Odd Combination (Figure 1(a))	4	4
	Valid Combination (Figure 1(a'))	2	2
2	Odd Combination (Figure 1(b))	5	8
	Valid Combination (Figure 1(b'))	2	4
3	Odd Combination (Figure 1(c))	9	12
	Valid Combination (Figure 1(c'))	2	4

where it moves a character in that direction from its current position. When it reaches either the end of a row or a column, it jumps to the other end of the corresponding row or column. The objects continues following this simple mobility model until the simulation is stopped. A crossing for a player is recorded when it moves to a different coloured region from its current region.

Table 2: Comparison of player crossings for both odd and valid aggregates.

Case	Description	Number of Players		
		1	5	10
1	Odd Combination (Figure 1(a))	5	24	46
	Valid Combination (Figure 1(a'))	2	11	18
2	Odd Combination (Figure 1(b))	9	51	86
	Valid Combination (Figure 1(b'))	5	21	46
3	Odd Combination (Figure 1(c))	18	78	138
	Valid Combination (Figure 1(c'))	7	24	51

3.2.2 Evaluation results

Table 2 summarises the simulation results for both odd and valid combinations. It can be seen in first case, that crossings for odd combination are almost twice the number of crossings for the valid combination. Case 2, has a similar outcome, however, the crossings for odd combination are slightly less than twice the number of crossings for valid combination. This is due to the player distribution, and the ratio between isolated spaces and exposed boundaries for both combinations. It can be seen in case 3, that when there are more isolated regions and exposed boundaries, there are more crossings. The crossings for odd case are almost three times the crossings for valid combinations. Overall, the simulation results revealed that odd cases greatly increases the crossings between the Sims in addition to the implementation complexity and communication overhead. In the next sections, we provide detailed illustrations of the basic and extended algorithms.

4 The extended aggregation algorithm

4.1 Limitations in basic algorithm

The basic aim of aggregation algorithm was to aggregate smaller regions into larger contiguous spaces for assignment. It initially takes regional grids of n×n dimensions as an input normally based on the split strategies of our scalable infrastructure. It repeatedly assigns different parts of the pre-processed space to additional Sims and it has to cope with varied shapes of spaces. In theory, the current aggregation algorithm should always yield valid combinations but in fact 'practically' it allows odd combinations for the non-square shaped grids obtained after the first split and assignment applied to a square grid. During implementation, it failed to discard odd cases in the following iterations. In other words, starting with a square grid, the first iteration determines valid contiguous spaces but in later iterations, when it is applied to non-square shaped worlds, it allows odd cases. Figure 2 illustrates these cases with the help of a simple square grid of nine regions (labelled A to I). The first iteration of aggregation algorithm divides this grid into two aggregates (colours are used to differentiate aggregates from each other) having A and B in the first and the rest of the regions in the second aggregate (see Figure 2(a)). However, when it is applied to the second aggregate (a 7-region world) in second iteration, it selects an aggregate comprises of region C and D, which are both isolated than each other (an obvious odd case), as shown in Figure 2(b). This is because the basic algorithm only uses SplitCapacity constraint but does not check explicitly the continuity constraint for the space comprises of smaller sub-regions. An extension to the current algorithm is presented in the next section to overcome these issues.

4.2 The extended algorithm

In each step of the aggregation process, an additional step is added to make it sure that both prospective aggregates produce valid contiguous spaces. This additional step explicitly use a flood fill algorithm to check continuity in the aggregated spaces. We use flood fill algorithm that spread in four ways as the one that spreads in eight ways consider the diagonals which are major source of odd combinations. Flood fill algorithms are normally used in bucket fill al-

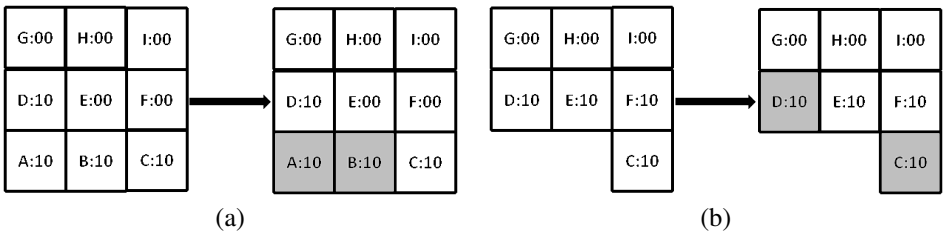

Figure 2: Illustrating limitations in the basic aggregation algorithm: (a) A valid outcome for a square grid of 9 regions; (b) An invalid outcome for a 7-region world.

gorithms of paint programmes, and they are employed in board games such as Go and Minesweeper [30]. In each step, when the possible aggregates are determined by the aggregation algorithm, it checks these aggregates against the continuity constraint, and reject them when any of them are not constituting a valid contiguous space. The extended algorithm has the capability to determine and exclude odd cases against any size and shape of a given space.

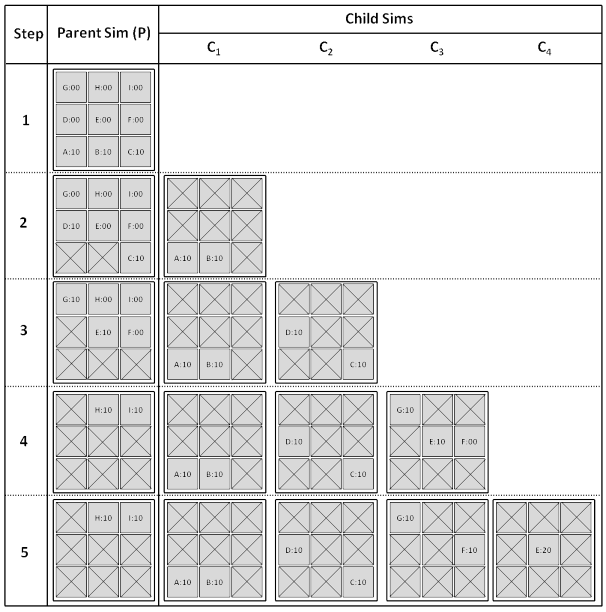

Figure 3: Expanding a 9-region world with the basic aggregation algorithm.

4.3 Illustration and comparison of basic and extended algorithms

In this section, we illustrate the limitations in basic aggregation algorithm and the worth of extended algorithm eliminating issues in the current algorithm with simple player distributions. These example illustrations use a SplitCapacity of 40 players and applies the aggregation strategies to Bottom Left (BL) and Bottom Right (BR) against a 9-region world in a grid form. This article is illustrating only the limitations of current algorithm and it is not demonstrating the aggregation strategies which are presented in detail in [7, 9]. Figure 3 illustrates odd cases allowed by basic algorithm whose equivalent valid combinations which are obtained using the extended algorithm are presented in Figure 4. The partial steps (showing expansion up-to 4 child Sims) shown in these figures are highlighting important points during split and assignment processes. A Sim includes the number of players in each named region that it hosts, and the regions hosted by other Sims are crossed with respect to this Sim.

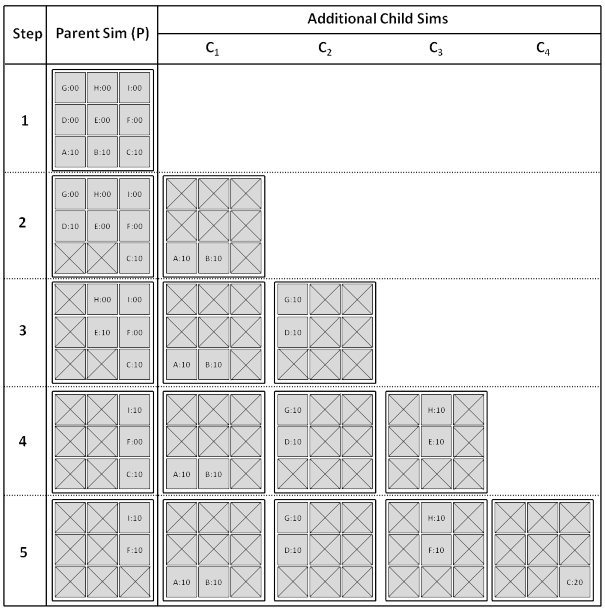

Figure 4: Expanding a 9-region world with the extended aggregation algorithm.

Figure 3, step-1, shows that the parent Sim is initially hosting the whole space comprises of nine regions. In step 2, the space is divided into two valid contiguous groups of regions and then the algorithm assigns the aggregate comprises of region A and B to child Sim C_1. However, the remaining steps assign odd combinations such as in, step 3, the parent Sim transfers region C and D to C_2. Similarly, the parent Sim in step 4, assigns an aggregate of region E, F and G to Child C_3. Further, the child C_3 assign a valid combination to child C_4, but maintains itself a non-contiguous space, in step 5. It is important to note that only the RR strategy of the first root obtained the aggregates assigned in Figure 3.

Figure 4 illustrates the extended aggregation algorithm for exactly the same player distribution used in Figure 3. It is obvious that the extended algorithm strictly allows only valid continuous spaces. The algorithm, in step-2, divides the space into two bigger spaces and assigns the aggregate comprises of region A and B to child C_1, but after verifying the other aggregate being a contiguous one as well. It can be noted in step 3, that the extended algorithm determines the non-contiguous aggregate comprises of C and D to be an odd case and it is skipped by the algorithm to be an acceptable aggregate. The RR strategy was unable to determine further aggregates for a 7-region world at this stage and the algorithm, therefore, applied the CC strategy, which determined a valid aggregate comprises of D and G, in step-3 and assigned it to C_2. Step-4, rejected the assignment of aggregate comprises of region C and E but instead assigned a contiguous space made of E and H to child C_3 determined using the CC strategy. Region C is then assigned during step 5 to child C_4.

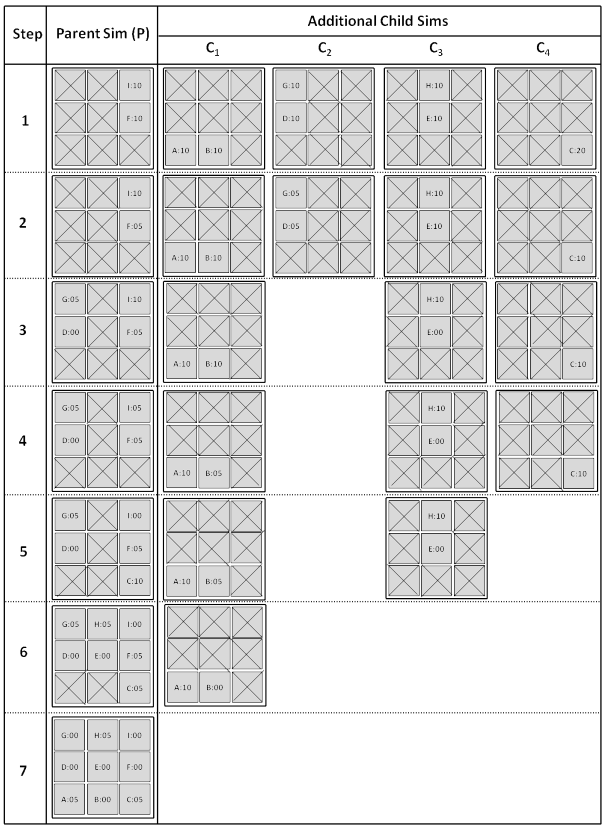

Figure 5: Contracting a 9-region world with the basic PntMrg strategy.

merging preferences either with a parent or a sibling Sim. Both a child and the parent have the capability to determine if a merge operation to be initiated when they notice a decrease in their current capacities but the merging process is always initiated by a child Sim in current implementation. Both the strategies use a MergeCapacity constraint to initiate a merge. Despite the status of a Sim being parent or a child, it first determines, if the cumulative load of both the Sims to combine their load, is less than or equal to the MergeCapacity. On satisfying this condition, the child Sim assigns its complete load (both content and players) to the participating Sim and releases itself.

Both strategies have a flaw (similar to the one for split discussed earlier) that they allow odd combinations while integrating the load which violate the basic goals of our work. The MergeCapacity value of 20 players is considered in this work, a much smaller value to avoid immediate splits.

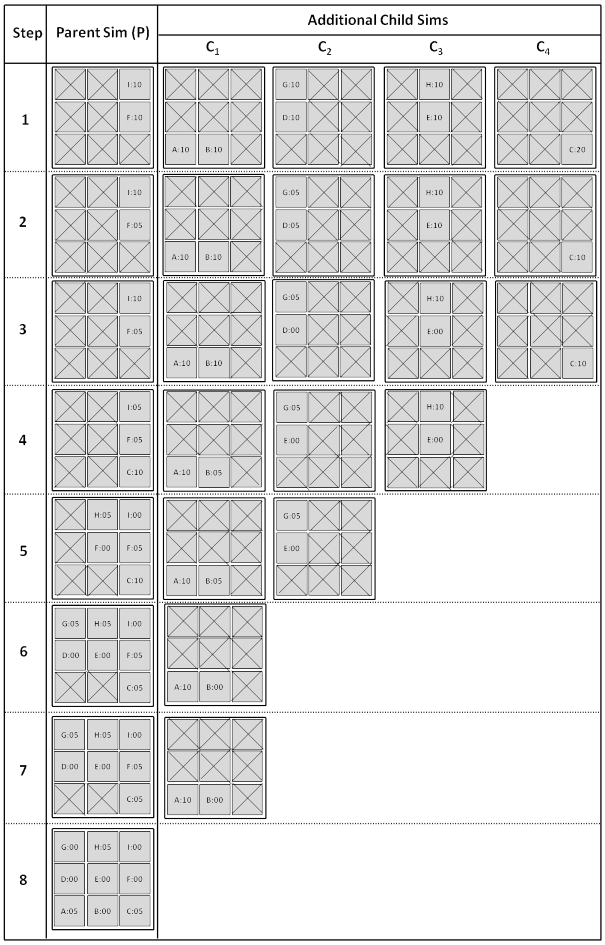

Figure 6: Contracting a 9-region world with the extended PntMrg strategy.

5 The extended merging algorithms

5.1 Limitations of the basic algorithms

The current merging process provides two algorithms (implementing a PntMrg and a ChMrg strategy) that differ by

5.2 The extended algorithms

To avoid assigning non-contiguous spaces, this work also explicitly incorporate an additional step which determines

that either a combined space of two Sims are constituting a contiguous space or not using a flood fill algorithm in addition to the MergeCapacity constraint. A merge is only allowed, if it passes through the continuity check, otherwise, the merge is rejected. This model might use more than required number of Sims for a little longer but it achieves the benefits of assigning contiguous spaces to different Sims.

5.3 Illustration and comparison of basic and extended algorithms

5.3.1 The Parent Merge (PntMrg) strategy

Figure 5 illustrates the basic PntMrg procedure. No merge is permitted with the parent Sim in initial two steps, because the cumulative load in each case is more than the MergeCapacity. However, it is clear in step 2 that child Sims C_2 and C_4 satisfies the merge condition but it is not allowed in PntMrg strategy. Child C_2 integrates its load with the parent Sim during step 3. However, it can be seen that this merge results-in a space comprises of two isolated spaces. In step 4, no merge is allowed though a merge is possible among child Sims C_3 and C_4. The aggregated space after the integration of space maintained by C_4 with the parent Sim in step 5 also gives two isolated sets of regions. The PntMrg strategy potentially holds more resources than required for longer time compared with the ChMrg strategy which tries to overcome this issue.

Figure 6 illustrates the extended PntMrg strategy highlighting the avoidance of odd cases allowed by the basic algorithm as shown earlier in Figure 5. No merge was permitted during the initial three steps. Capacity constraint did not allow merging of child Sims C_1 and C_4 with the parent Sim. The merge between child Sim C_2 and parent Sim at step-3 was rejected due to the continuity constraint. Child C_4 returned its space to the parent Sim at step-4. Child Sims C_3, C_2 and C_1 integrated their load with parent Sim at step 5, 6 and 8 correspondingly. Figure 6 shows that the extended algorithm keep resources for more time than the basic algorithm as illustrated in Figure 5.

5.3.2 The Child Merge (ChMrg) strategy

Figure 7 illustrates the basic ChMrg procedure. No Merge was allowed in step-1, due to the MergeCapacity constraint. However, C_4 was released after merging its load with C_2 at step-2 (a case which was rejected by the PntMrg strategy) but constituting an obvious odd case. Step-3 and 5 obtained valid combinations (the first between the parent and C_3, and the second one between C_1 and C_2) of space after merging, however, it was demonstrated that the basic ChMrg merging strategy allows odd combination.

Figure 8 illustrates the extended ChMrg algorithm that consider both the capacity and continuity constraints for initiating a merge. It always yields contiguous spaces and, therefore, rejected, a merge between child Sims C_2 and C_4. It improves over PntMrg strategy in a sense that it merges quicker by considering the child Sims as in step 4, where

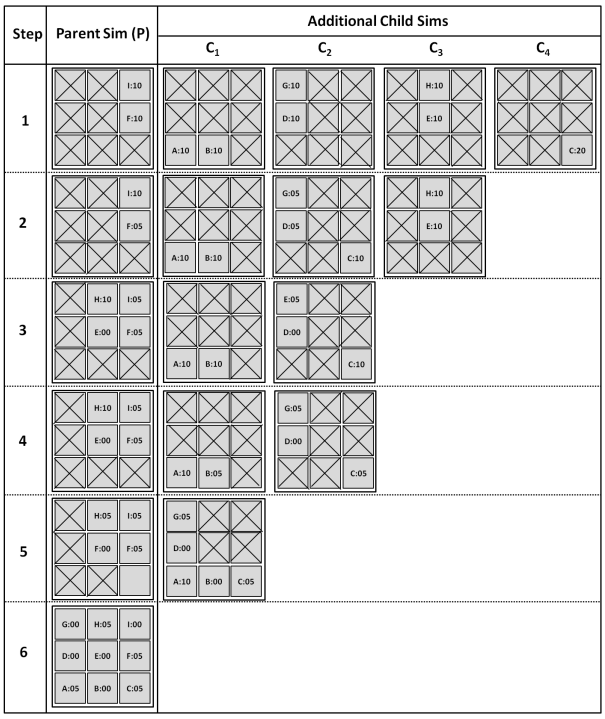

Figure 7: Contracting a 9-region world with the basic ChMrg Strategy.

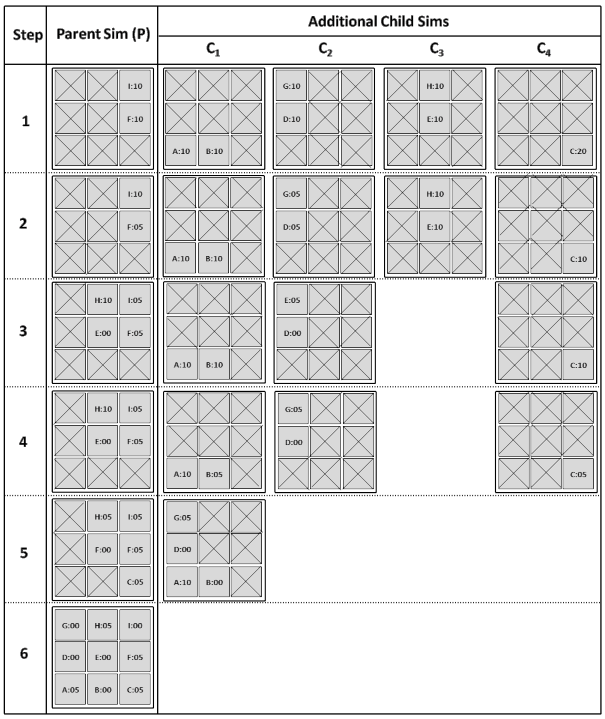

Figure 8: Contracting a 9-region world with the extended ChMrg strategy.

two integrations happened, one between the parent and C_4 and the other between C_1 and C_2. However, it potentially transfers the content and players multiple times which might degrade the overall system performance.

5.3.3 Discussion

The merging strategies (both PntMrg and ChMrg) demonstrated in this work have worth and limitations. Both of them ultimately return the whole world back to the parent Sim. Normally, a merge operation is initiated when player capacity is not high. The PntMrg strategy is simple but takes more time and holds resources for longer than the ChMrg strategy. The ChMrg strategy copes with the issues in PntMrg strategy and release resources much quicker. However, the ChMrg strategy potentially transfers regions (both content and players) between Sims multiple times and it brings a bad experience to the users. We have demonstrated both the strategies, and they could be adopted according to requirements. However, the basic strategies were unable to avoid odd cases. Odd combinations are rejected by both the extended strategies. To manage bigger worlds and the un-predictable nature of users, we suggest using ChMrg strategy as PntMrg might be blocked for longer. However, both have the potential to cope with resource under-utilisation issues. Further details on this are beyond the scope of this article and interested readers may read our detailed work on this in [6].

6 Conclusion

In this article, we presented the extended aggregation and merging processes, to cope with the limitations in basic versions of these mechanisms. It provided an overview of our scalable infrastructure comprises of splitting, merging and load distribution algorithms in comparison with other mechanisms found in the Literature. It examined current and extended operations for both the aggregation and merging, and provided a justification for the continuity model in addition to SplitCapacity and MergeCapacity in their corresponding operations. The extended operations have potential of getting aggregation and merging robustly and they are illustrated with some simple examples from the results obtained from our prototype for scalable virtual worlds.

References

[1] Pekka Alahuhta, Emma Nordbck, Anu Sivunen, and Teemu Surakka. Fostering team creativity in virtual worlds. *Journal For Virtual Worlds Research*, 7(3), 2014.

[2] Rajesh Krishna Balan, Maria Ebling, Paul Castro, and Archan Misra. Matrix: Adaptive Middleware for Distributed Multiplayer Games. volume 3790/2005 of *Lecture Notes in Computer Science*, pages 390–400. Springer Berlin/Heidelberg, 2005.

[3] A. M. Burlamaqui, M. A. M.S. Oliveira, A. M. G. Goncalves, G. Lemos, and J. C. De Oliveira. A Scalable Hierarchical Architecture for Large Scale Multi User Virtual Environments. In *IEEE International Conference on Virtual Environment, Human Computer Interfaces and Measurement Systems*, pages 114–119, 2006.

[4] Luther Chan, James Yong, Jiaqiang Bai, Ben Leong, and Raymond Tan. Hydra: A Massively-Multiplayer Peer-to-Peer Architecture for the Game Developer. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, NetGames '07, pages 37–42, New York, NY, USA, 2007. ACM.

[5] F. Chang, C.M. Bowman, and W. Feng. XPU: A Distributed Architecture for Metaverses. Technical report, Department of Computer Science, Portland State University, 2010. Technical Report 10-04.

[6] Umar Farooq. *The Design of a Contemporary Infrastructure for Scalable and Consistent Virtual Worlds*. PhD thesis, School of Computing Sciences - University of East Anglia, 2012.

[7] Umar Farooq and John Glauert. ARA: An Aggregate Region Assignment Algorithm for Resource Minimisation and Load Distribution in Virtual Worlds. In *NDT '09: Proceedings of the first IEEE International Conference on Networked Digital Technologies*, pages 404–410, 2009.

[8] Umar Farooq and John Glauert. Joint Hierarchical Nodes based User Management (JoHNUM) Infrastructure for the Development of Scalable and Consistent Virtual Worlds. In *DS-RT '09: Proceedings of the 13th IEEE/ACM Symposium on Distributed Simulation and Real-Time Applications*, pages 105–112, Washington, DC, USA, 2009. IEEE Computer Society.

[9] Umar Farooq and John Glauert. A Dynamic Load Distribution Algorithm for Virtual Worlds. *Journal of Digital Information Management*, 8(3):181–189, June 2010.

[10] Umar Farooq and John Glauert. Scalable Virtual Worlds: An Extension to the OpenSim Architecture. In *ICCNIT '11: Proceedings of the IEEE International Conference on Computer Networks and Information Technology*, pages 29–34, 2011.

[11] Umar Farooq and John Glauert. Faster dynamic spatial partitioning in opensimulator. *Virtual Reality*, 21(4):193–202, Nov 2017.

[12] Umar Farooq and John Glauert. Integrating dynamic scalability into the opensimulator framework. *Simulation Modelling Practice and Theory*, 72(2017):118–130, 2017.

[13] Paul A. Fishwick. An introduction to opensimulator and virtual environment agent-based m&s applications. In *Winter Simulation Conference*, WSC '09, pages 177–183. Winter Simulation Conference, 2009.

[14] R. M. Fujimoto, K.S. Perumalla, and G.F. Riley. *Network Simulation*. Synthesis lectures on communication networks. Morgan & Claypool Publishers, 2007.

[15] N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, and W. White. Scalability for Virtual Worlds. In *Proceedings of the 2009 IEEE International Conference on Data Engineering (ICDE '09)*, pages 1311–1314, 2009.

[16] Marko Hakonen and Petra Bosch Sijtsema. Virtual worlds enabling distributed collaboration. *Journal For Virtual Worlds Research*, 7(3), 2014.

[17] IDC. Butterfly.net: Powering next generation gaming with on-demand computing. Technical report, IBM: An IDC e-Business Case Study, 2004.

[18] Beob Kyun Kim and Kang Soo You. A Hierarchical Map Partition Method in MMORPG based on Virtual Map. In *Frontiers of High Performance Computing and Networking - ISPA 2006 Workshops*, volume 4331/2006 of *Lecture Notes in Computer Science*, pages 813–822. Springer Berlin/Heidelberg, 2006.

[19] Dan Lake, Mic Bowman, and Huaiyu Liu. 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, pages 19:1–19:6, Piscataway, NJ, USA, 2010. IEEE Press.

[20] Charles J. Lesko and Yolanda A. Hollingsworth. Architecting scalable academic virtual world grids: A case utilizing opensimulator. *Journal For Virtual Worlds Research*, 6(1), 2013.

[21] H. Liu and M. Bowman. Scale Virtual Worlds through Dynamic Load Balancing. In *DS-RT '10: Proceedings of the 2010 14th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 43–52, Washington, DC, USA, 2010. IEEE Computer Society.

[22] Qingqi Long, Jie Lin, and Zhixun Sun. Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations. *Simulation Modelling Practice and Theory*, 19(4):1021 – 1034, 2011. ¡ce:title¿Sustainable Energy and Environmental Protection SEEP2009¡/ce:title¿.

[23] John C. S. Lui and M. F. Chan. An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems. *IEEE Transaction on Parallel and Distribution Systems*, 13(3):193–211, 2002.

[24] Thomas M. Malaby. *Making Virtual Worlds: Linden Lab and Second Life*. Cornell University Press, Ithaca, United States, first edition, June 2009.

[25] Beatrice Ng, Antonio Si, Rynson W. H. Lau, and Frederick Li. A Multi-server Architecture for Distributed Virtual Walkthrough. In *ACM Symposium on Virtual Reality Software and Technology*, pages 163–170. ACM New York, NY, USA, 2002.

[26] K. Prasetya and Z. D. Wu. Performance Analysis of Game World Partitioning Methods for Multiplayer Mobile Gaming. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 72–77, New York, NY, USA, 2008. ACM.

[27] Michael Rymaszewski, Wagner James Au, Mark Wallace, Catherine Winters, Cory Ondrejka, Benjamin Batstone-Cunningham, and Philip Rosedale. *Second Life: The Official Guide*. Wiley Publishing, Hoboken, New Jersey, December 2006.

[28] Shervin Shirmohammadi, Ihab Kazem, Dewan Tanvir Ahmed, Madeh El-Badaoui, and Jauvane C. De Oliveira. A Visibility-Driven Approach for Zone Management in Simulations. *Simulation*, 84(5):215–229, 2008.

[29] Matteo Varvello, Fabio Picconi, Christophe Diot, and Ernst Biersack. Is There Life in Second Life? In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 1:1–1:12, New York, NY, USA, 2008. ACM.

[30] Wiki. Flood fill algorithm. `http://en.wikipedia.org/wiki/Flood_fill`, 2016. Accessed: December, 2016.