Scaling knowledge based virtual worlds: An efficiency gain through multi-core architectures

DOI : 10.36909/jer.10973

Umar Farooq[*], Zohra Israr[*], Ihsan Rabbi[*], Kashif Zia[**]

[*]Department of Computer Science, University of Science and Technology Bannu, Pakistan.
[**]Faculty of Computing and Information Technology, Sohar University, Oman.

[*]Email: ihsanrabbi@gmail.com; Corresponding Author.

ABSTRACT

This paper investigates the impact of sequential content load algorithm on regional transfer in OpenSimulator framework. It, then, presents an abstract parallel content load algorithm, which is implemented using the basic parallel for and chunk partitioning constructs to exploit the parallelism of multi-core architectures. This work used a set of five example virtual worlds content to investigate the effectiveness of the proposed work and its impact on the reallocation process of regions in OpenSimulator. Simulation results revealed that the implementation using parallel for reduced the timings by 47% to 57% while the implementation using chunk partitioning minimized it by 63% to 72%. Time impact on regional transfer was reduced by 39% to 51% using parallel for while 52% to 66% using chunk partitioning.

Key words: scalability; virtual worlds; OpenSimulator; multi-core architecture; efficiency.

INTRODUCTION

Virtual Worlds (VWs) are virtual environments featuring unique characteristics such as coherence, interaction, collaboration, persistence and being social in nature (Hakonen & Bosch, 2014). They have recently been emerged as an innovation and knowledge management platform due to their social nature. These worlds are interactive and collaborative in nature, and support social aspects necessary for creating and sharing knowledge (Bredl et al., 2012). These worlds have gained popularity in development and research communities due to their uniqueness and the provision of rich appealing content (Yilmaz et al., 2014). The huge popularity of VWs has introduced new dimensions of knowledge acquisition and learning over the Internet (Bredl et al., 2012). Second Life (SL) (Linden Research, Inc., 2021) and OpenSimulator (OSim) (OSim-Wiki, 2021) are the well-known VWs available today. VWs are different than games and they allow players to develop their own content which persist even if the players go off-line (IDC, 2004; Farooq & Glauert, 2017a). The users, represented as digital characters called avatars, participate in social interaction and collaborative activities with each other’s for both fun and implementing creative ideas (Alahuhta et al., 2014). Avatars are knowledgeable workers and within VWs, they involve better in IT related tasks for interaction and cooperation. The flexibility and scalability traits make these environments suitable for both constructivist and connectivistic learning (Bredl et al., 2012). VWs that embed knowledge workers are practical platforms for user centered knowledge acquisition and cooperation. They are suitable and affordable means to effective learning and knowledge dissemination (Ata & Orhan, 2013). Sox et al., (2014) extended the meeting and business event competency standards by incorporating the best practices, possibilities and barriers of using virtual and hybrid meetings learnt from industry personnel. Equipped with a knowledge management approach, businesses can get new market knowledge and utilize it to compete in the competitive market.

Due to the tremendous popularity and response from the end users, VWs run over grid infrastructures to scale well (Behnke, 2020). The current grid infrastructures use static spatial partitioning methods for an improved interactive experience. They, however, result in poor utilization of resources. To overcome these issues, dynamic strategies must be introduced to provide resources purely based on load. The dynamic spatial partitioning, however, is considered a very expensive operation as it transfers both content and players. Based on an extended OSim framework (Farooq & Glauert, 2017b), a dynamic grid infrastructure was proposed for which a working prototype was developed using the OSim framework (Farooq & Glauert, 2017a). The modular structure of OSim framework and the development of optimal strategies for region removal greatly assisted the developers in making dynamic spatial partitioning a considerable approach to scale VWs (Farooq & Glauert, 2017a). For content transfer, they adopted the OSim back-up mechanism, called, the OpenSimulator Archive (OAR) functionality. The current implementation, however, use a sequential approach to set-up a scene from an OAR repository and load the components (including assets, terrain, regional settings, objects and parcels) in order. Therefore, it adds a considerable amount of time to the content unavailability time (Farooq & Glauert, 2017a). Faster methods (briefly introduced in (Farooq et al., 2018)) are needed to reduce the impact of content transfer using the OAR functionality.

This paper briefly explores the existing multi-core architectures and C# (the native language of OSim framework) parallel programming constructs and, then, presents an abstract parallel algorithm for loading regional content. It implements the proposed algorithm using two parallel constructs and provides their comparison with the sequential approach. Simulation results suggest clear improvements of the proposed parallel algorithm over the existing sequential algorithm. This study is different than others studies as it pursues a dynamic scalability framework, which is considered, expensive in terms of time and, thus, usually avoided. This paper is organized as follows. The INTRODUCTION section explores the research area undertaken in this work. The complete background, and motivation and goals of this work are outlined in BACKGROUND AND MOTIVATION section. It explores the OSim framework in general and its OAR functionality in particular. It also provides the existing mechanisms to manage scalable VWs. The INVESTIGATING THE SEQUENTIAL ALGORITHM section presents the investigation results of sequential content load algorithm and its impact on the overall regional transfer process. The parallel multi-core architectures and parallel programming constructs of C# language are provided in MULTI-CORE ARCHITECTURES AND C# PARALLEL PROGRAMMING section. The PROPOSED PARALLEL ALGORITHM section introduces the proposed parallel algorithm. Evaluation results for the current implementations of the proposed parallel algorithm and their comparison with the sequential approach are also provided in this section. It also presents the impact of proposed algorithm on regional transfers. The CONCLUSION section provides the conclusion while FUTURE DIRECTIONS section lists future directions based on this work.

BACKGROUND AND MOTIVATION

OpenSimulator (OSim) is an open source 3D VW development framework (OSim-Wiki, 2021). It is written in C# language and it can run on both Windows and Linux machines. It is capable of creating environments as small as one Simulator (Sim) to as large as thousands of Sims (Farooq & Glauert, 2017b). It is cost effective and very flexible for creating, archiving, and sharing virtual content compared with other platforms (Behnke, 2020; OpenSim, 2021).

Region is a basic unit of named 256m × 256m area which is hosted by a single Sim. A single Sim, however, can possibly host arbitrary number of regions (Farooq et al., 2018). Each region has an associated scene that represents not only the content and avatars but also all the relevant computation and communication concerns (Liu & Bowman, 2010; Farooq & Glauert, 2017a). Regions are organized by a 2D Grid. Each region in a Grid has unique location represented by X and Y co-ordinates. Grid manages different issues across regions. OSim allows multiple client viewers (such as Phoenix, Firestrom, Imprudence and Hippo), and protocols to access VWs (OSim-Wiki, 2021; Alahuhta et al., 2014). User, Grid, Asset, Inventory, and Messaging *(UGAIM)* Services are used to provide interaction between a region and viewers (Hakonen & Bosch, 2014).

OSim can run possibly in Standalone or Grid mode (Farooq et al., 2018; OSim-Wiki, 2021). They are differentiated in terms of the place from where a Sim gets its UGAIM services. In standalone mode, a single process (opensim.exe) provides a region Sim and UGAIM services. OSim uses SQLite when configured in standalone mode (OpenSim, 2021). Standalone mode is very simple to configure but very limited in scope. Different aspects of simulation in grid mode are distributed among different processes which can possibly held on different machines. Currently, however, the UGAIM services are available as a single data services module (Robust.exe) while the region Sim is implemented as a separate process (opensim.exe). It allows running arbitrary number of region Sims each possibly on a different machine. Grid mode utilizes MySQL to host the data of different services (Farooq & Glauert, 2017b). Grid service not only support regions hosted by different Sims to know each other but also facilitate players' movement between regions hosted by different Sims (Hakonen & Bosch, 2014; Farooq & Glauert, 2017a; OSim-Wiki, 2021).

OpenSimulator Archive (OAR) functionality is used to back-up the content of the whole region in a named file, called OAR, which can be later on, uploaded to a newly created region on a completely different Sim. OAR files are much advanced and they not only store and load scene data but the complete terrain, textures, assets, objects and all relevant items attached to a scene (Farooq & Glauert, 2017a; Farooq & Glauert, 2017b). An OAR file comprises of different files and folders to hold the complete content of a region. These files and folders include an archive file (archive.xml) and other components that are assets, objects, terrain, region settings, and parcels. The size of an OAR file depends on the number of assets, and scene objects and their complexity (OpenSim, 2021; Farooq et al., 2018).

These files can also be used to share the content with others and they can be loaded to a completely new region on a different Sim. OAR functionality provides the following two commands to manage regional back-ups (Farooq & Glauert, 2017b; OSim-Wiki, 2021):

- **Save OAR (SOAR):** command manages back-ups by storing the content of an OSim region into a named OAR file.
- **Load OAR (LOAR):** command loads the content provided in an OAR file to the current region. The basic command deletes the existing objects in a scene before loading the content from OAR file. The merge option, however, merges the content of OAR file content to the existing objects.

This work concentrated on developing an extended parallel content load algorithm, as the basic sequential content load algorithm was identified as an activity having huge contribution towards total re-allocation process of a region, in terms of time. Table 1 provides the content load timings for the sequential approach while Table 2 presents the impact of sequential load timings on the re-allocation process of a region.

1. Existing Scalability Mechanisms

Grid infrastructures are the most favorable to run scalable applications such as VWs (Behnke, 2020; Farooq & Glauert, 2017b) due to their potential benefits over the centralized, clustered, and Peer-to-Peer (P2P) infrastructures (Spieldenner et al., 2018; Abdulazeez & Rhalibi, 2018). The newly emerged cloud infrastructures are capable of meeting the computational and communication needs of ever growing requirements of VW environments (Kim, 2020). Cloud resources are, however, much costly than dedicated servers (such as those part of a grid) for running VWs which are up and running all the time (Rogers, 2021). Grids, therefore, not only provide scalable, dynamic, mature, and resilient infrastructures but also cost effective solutions to run scalable applications such as VWs (Behnke, 2020; Farooq & Glauert, 2017a).

SL (Linden Research, Inc., 2021) and OSim (OSim-Wiki, 2021) are hosted over dedicated grids, called, SL Grid (SLG) (Farooq & Glauert, 2017b) and OSim Grid (OSimG) (Osgrid.org Inc. 2021; Farooq & Glauert, 2017a), correspondingly. The OSimG is more flexible than the SLG and it allows an arbitrary number of regions to host per Sim. They both, however, suffer from significant load imbalance due to the static assignment of space in a flat orientation. Furthermore, they are unable to cope with both the under-provision and over-provision of resources. They, therefore, degrade the interactive user experience. Cloud implementation of grids also suffers from the above mentioned issues. Flat dynamic mechanism with a wide range of local, global and adaptive strategies are developed to overcome the issues in flat static mechanisms (Farooq & Glauert, 2017b; Farooq & Glauert, 2017a). They, however, degrade the interactive user experience as the migration of users incurs additional processing burdens. Dynamic hierarchical approaches are more flexible and scalable than the flat mechanisms, however, they impose no restrictions on the levels in a resource management tree (Farooq et al., 2018; Farooq & Glauert, 2017b). They are, therefore, more complex and degrade the interactive user experience.

The use of simulation centric architecture by the current VWs including the OSim framework greatly restrict their capabilities in terms of players and objects. The ScienceSim project has contributed much to the enhancement of OSim code for performance and scalability. It uses the basic OSim framework but improves its performance by using Distributed Scene Graph (DSG) architecture, which detaches data from the scene (Liu & Bowman, 2010; Farooq et al., 2018). Srikata is another similar architecture, which reduces network load of the main server with the help of an independent Content Delivery Network (CDN). CDN performs synchronization of the static and less frequently updated data (Elfizar et al., 2019). DSG and Srikata increase the number of objects, they manage, however, the capacity of VWs in both, in terms of objects and players is still limited. To further scale VW environments, another Sim detachment method called: One Process for One Object (1P1O) architecture (Elfizar et al., 2019) was proposed. The authors of 1P1O claim that this architecture is more scalable than DSG and Srikata. However, it is yet in its initial stages and not much explored. In general, DSG, Srikata and 1P1O involve additional layers and they, therefore, have an increased level of complexity, which potentially introduces longer delays. Furthermore, they target different aspects than spatial partitioning to scale the existing environments.

In order to support hotspot scenarios in an adequate manner, Behnke (Behnke, 2020) proposed a technique that decouples state retention from processing capabilities. The proposed mechanism enables processing a given players' intensive locality by a set of game servers while maintaining a single DVE instance. The proposed infrastructure is, however, more complex due to an additional layer for game servers. Further, it targets only one dimension (the number of players) to scale. A networked platform called, DiVE, was developed for developing highly scalable 3D virtual presences by integrating the user friendliness of VWs while scale and optimization of game environments (Prendinger et al., 2018). It optimizes zone based interest management technique for an improved scalability. However, similar filtering mechanisms are commonly used in existing VW platforms.

Load Balancing in OSim framework using spatial partitioning was investigated only once by a project called Load Balancer (Farooq et al., 2018; Farooq & Glauert, 2017b). However, it implemented the game specific techniques, which resulted in a degraded interactive experience of VW users. Furthermore, it is no longer maintained.

To cope with the issues in static and dynamic systems described above, a dynamic hierarchical framework based on expansion and contraction models was proposed (Farooq & Glauert, 2017b). For transferring regional content, it adopted using the OAR functionality that is capable of setting-up the whole scene while hiding the complexities involved in a transfer. A prototype of the proposed mechanism was developed using the modular structure of OSim framework and its basic components (Farooq & Glauert, 2017b). Extended strategies were developed for removing regions from a Sim. It greatly improved the re-allocation process, however, the current content load algorithm, which follows a sequential approach, adds a huge amount of time to the process (Farooq & Glauert, 2017a; Farooq et al., 2018). To improve user interactive experience, faster methods for this activity must be explored.

This work investigates the existing content load algorithm and its impact on regional re-allocation process. It, then, presents a detailed analysis of the implementation of the proposed abstract parallel content load algorithm (introduced briefly in (Farooq et al., 2018) and its improved impact on the re-allocation process.

2. Motivation and Goals

The considerable amount of time consumed by the standard content load algorithm of OSim framework and its huge impact on regional transfers motivated us to develop an alternate parallel version of the sequential algorithm. It provides not only a valuable contribution by developing a faster content load algorithm but also improves the re-allocation process in dynamic scalable OSim environments.

INVESTIGATING THE SEQUENTIAL ALGORITHM

This section explores the sequential algorithm for loading content to a region, currently available in OSim framework and investigates the time, it consumes in loading regional content. It, then, highlights its impact on the re-allocation process.

1. The Sequential LOAR Algorithm

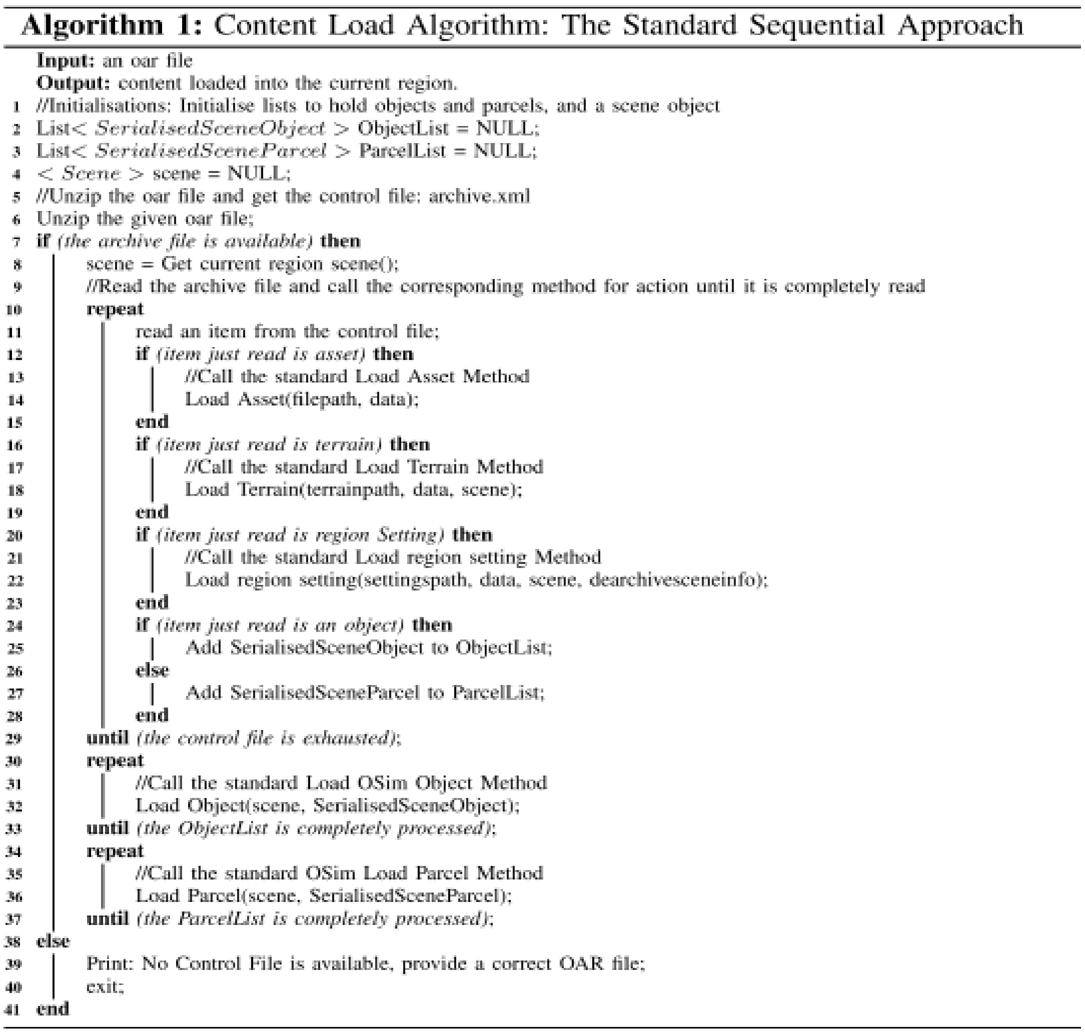
This section presents the sequential algorithm (see Algorithm 1) for loading regional content from an OAR file, provided as an input, using the OAR functionality. Wherever, it was possible, we have used the existing methods of OSim framework. These methods are used at abstract level and may not be following the exact form of the OSim methods. Similarly, further details of individual activities are not provided. However, descriptions are added to these methods for understanding purposes. The algorithm first extracts the OAR file and, then, checks the type of each item (which could possibly be an asset, a terrain, region settings, an object or a parcel) it reads from the centralized control file. It determines the scene from regional information provided of the archive file. Based on the type of each item, the algorithm initiates the corresponding method to manage loading that item.

The current content load algorithm, sequential in nature, is presented as algorithm 1. It takes an OAR file as an input. It first unzips the given file to get the content repository. The algorithm determines the archive control file (archive.xml) and if found, it reads one item (either the terrain, asset, object, parcel or region settings) at a time and, then, calls its corresponding method to handle the item just read. The main loop iterates sequentially through the complete file and it calls the load asset (by providing the file path and data as arguments), load terrain (that takes terrain path, data and scene as arguments) or load region settings (by providing settings path, data, scene and dearchivesceneinfo) to handle the assets, terrain, and regional settings. Apart from the above, it may read objects and parcels and add them to their corresponding lists.

Once the loop that iterates through the control file is terminated, the algorithm first goes through the ObjectList and add all the objects in it to the scene. It, then, adds all the parcels of the region (the ParcelList) to the scene.

2. Evaluation and Time Analysis

Farooq & Glauert (2017b) investigated three different kind of workloads (static, dynamic and example world), adopted from ScienceSim project, in terms of timings. The static workload was comprised of 15,000 individual unit Primitives (PRIMs) with no dynamism. The dynamic workload was comprised of 3000 individual PRIMs but with an attached script, which was rotating each PRIM and changing its color every 5 seconds. They also investigated example worlds content, in which the objects had simple to complex structures comprised of individual PRIMs. They revealed that it takes more time to load content with complex scenes compared with simple content even if they have dynamic factors. A large number of such content could be found in the Literature, which are free of cost available for experimentation and use. This work is, however, using a set of 5 small and medium size well known example worlds content described in Table 1 for experiments here. These content are described in terms of PRIMs, assets, objects and scripts.



Algorithm 1: Content Load Algorithm: The Standard Sequential Approach

```
Input: an oar file
Output: content loaded into the current region.
//Initialisations: Initialise lists to hold objects and parcels, and a scene object
List< $SerialisedSceneObject$ > ObjectList = NULL;
List< $SerialisedSceneParcel$ > ParcelList = NULL;
< $Scene$ > scene = NULL;
//Unzip the oar file and get the control file: archive.xml
Unzip the given oar file;
if (the archive file is available) then
    scene = Get current region scene();
    //Read the archive file and call the corresponding method for action until it is completely read
    repeat
        read an item from the control file;
        if (item just read is asset) then
            //Call the standard Load Asset Method
            Load Asset(filepath, data);
        end
        if (item just read is terrain) then
            //Call the standard Load Terrain Method
            Load Terrain(terrainpath, data, scene);
        end
        if (item just read is region Setting) then
            //Call the standard Load region setting Method
            Load region setting(settingspath, data, scene, dearchivesceneinfo);
        end
        if (item just read is an object) then
            Add SerialisedSceneObject to ObjectList;
        else
            Add SerialisedSceneParcel to ParcelList;
        end
    until (the control file is exhausted);
    repeat
        //Call the standard Load OSim Object Method
        Load Object(scene, SerialisedSceneObject);
    until (the ObjectList is completely processed);
    repeat
        //Call the standard OSim Load Parcel Method
        Load Parcel(scene, SerialisedSceneParcel);
    until (the ParcelList is completely processed);
else
    Print: No Control File is available, provide a correct OAR file;
    exit;
end
```

a. Environment

This work is investigated over a Core-i5 system with 2.4GHz processor and a total of 4GB RAM running Windows 7 operating system. The processor comprises of four cores. It is used to run an OSim instance in standalone mode with a single region. In start of each experiment, we create an empty region which is, then, loaded with one of the example worlds content shown in Table 1. Loading times for each content is recorded which are used not only for determining its impact on regional transfers but also for comparison purposes with the proposed algorithm.

Table 1 The content description and its load times using the existing sequential approach.

Virtual World Content	**Content Description**					**Time Analysis**
	File Size (MB)	**Prims (Num)**	**Assets (Num)**	**Objects (Num)**	**Scripts (Num)**	**Sequential Approach (Sec)**
IST OAR (ARL-STTC, 2020)	41.4	5793	779	1723	10	180
Alfea OAR (Artis, 2020)	41.0	2078	1303	2686	124	250
Epic-Citadel (Cuteulala, 2020)	36.1	516	464	510	35	177
OSim-openvce (Edin.Uni, 2020)	20.1	2129	388	97	133	59
GoneCity (Haan, 2020)	12.4	4983	167	2683	50	65

b. Experimental Results

To get fair responses, an experiment against each content was repeated five times. Table 1 provides average results of the experiments conducted for the example worlds content considered in this work. The results for loading content are self-explanatory. For example: minimum time (59 seconds) was needed to load OSim-openvce content (Edin.Uni, 2020) while Alfea (Artis, 2020) took the maximum time (250 seconds) among all the content considered in this work.

It is witnessed that generally with increase in file size, the time to load content increases. This however, is not developing a common pattern, as similar sizes have different timings. Even there are instances, where the content load time for smaller size was more than the time for larger sizes. This, therefore, suggests that it depends on the complexity of the objects in the content. More investigations are needed, however, to develop a generic prediction model regarding content load timings.

3. The Impact of Sequential Transfer on Re-allocation Process

This work improves one activity, the time taken by the content load algorithm, of the scalability framework developed by Farooq & Glauert (2017b). This framework uses a five step process during both the expansion and contraction phases, when the transferring region has no players. However, it follows a seven step process as it includes two steps for transferring players in the region to transit region from the actual region and, then, sending them back from transit to actual region after its re-allocation. This work uses regions only with content to determine the contribution of content load algorithm towards the total time required for re-allocating a region.

a. Region Removal Strategy

This work adopts the optimal strategy developed by Farooq & Glauert (2017b) instead of using the basic standard methods for closing or deleting a region, who had huge time requirements. The optimal strategy which extends the basic close region, uses OAR functionality to bring back up-to-date scene to a region after the successful re-allocation of region. It disables the persistence step before closing a region. It reduces the time needed to remove a region from the Sim to a few seconds. Further, details on this can be found in Farooq & Glauert (2017a).

b. Time Parameters

The following parameters (adopted from Farooq & Glauert (2017a)) are used to represent timing for different activities during the experiments in this work:

1. Store Content Time (T_{StCon}) is the time needed to store regional content in an OAR file.
2. Remove Region Time (T_{RemReg}) is the time needed to remove a region from the Sim using the extended close region method.
3. Create Region Time (T_{CreReg}) is the time for creating a region on the destination Sim.
4. Load Content Time (T_{LdCon}) is the time needed to load content from an OAR file.
5. Region Transfer Time (T_{RegTfr}) is the total accumulated time of all activities involved

in a regional transfer.

Table 2 Time statistics of sequential content load algorithm for various worlds content.

Workload	**Time Statistics**				
	Individual				**Accumulated**
	***TStCon* (Sec)**	***TRemReg* (Sec)**	***TCreReg* (Sec)**	***TLdCon* (Sec)**	***TRegTfr* (Sec)**
IST OAR (ARL-STTC, 2020)	9	5	2	180	196
Alfea OAR (Artis, 2020)	11	7	2	250	270
Epic-Citadel (Cuteulala, 2020)	8	4	3	177	192
OSim-openvce (Edin.Uni, 2020)	6	3	3	59	71
GoneCity (Haan, 2020)	4	2	2	65	73

c. Evaluation Results and Discussion

Table 2 provides timings for individual activities as well as accumulated response for relocating a region using sequential approach. The simulation results revealed that for the content used in this work, the content load algorithm contributes at least 84% time in total time taken by the re-allocation process. Specifically, the IST content took a total time of 196 seconds in loading the content after relocating the region, in which 180 seconds were contributed by the content load itself. It means that 91% time was consumed by the content load algorithm due to its sequential nature. The same was observed for other content named Alfea, Epic-Citadel, OSim-openvce and GoneCity, where the content load process took 92%, 90%, 84%, and 89% of the total re-allocation time, in given order.

In simple, these results suggests that the content load has a huge contribution towards the reallocation process. To reduce the content unavailability time, this factor must be reduced as currently this is the costly operation in terms of time.

One possible ways of reducing this impact is the use of parallel processing. Since, most of the current systems are based on multi-core architectures, whose parallelism can only be exploited if parallel programs are written for them. The next section is devoted to explore multi-core architectures and C# (the native language of OSim framework) parallel programming constructs. It is believed that further performance gain in terms of reduced re-allocation process would greatly help making dynamic infrastructures a favorable choice. The PROPOSED PARALLEL ALGORITHM section presents the proposed parallel content load algorithm and compares it with the sequential approach.

MULTI-CORE ARCHITECTURES AND C# PARALLEL PROGRAMMING

The current multi-core processors vary in terms of number of cores per chip and the number of chips on a computer system. The well-known variations of them include the dual-core, quad-core and dual quad-core or octa-core systems. Dual core processors (such as Intel dual core) comprise of two cores on a single chip. Most of the current mobile computers (such as Core i-5) use two core processors but a four thread implementation for improved performance. Quad core processors (such as Intel Core i-5) comprise of four independent cores in a single physical processor. Octa Core/Dual Quad Core processors comprise of eight cores in a single physical CPU package. In Octa core, all the cores are organized in a single chip. However, in dual quad core, they are organized in two quad core chips (Shahvarani & Jacobsen, 2019).

Multi-core systems use the concept of multi-threading to run multiple tasks on different cores simultaneously with an emphasis on balancing the total workload for an improved performance. The sequential approach used by the standard computers (Harris & Harris, 2013) executes the instructions in a pre-defined order (Campbell et al., 2010). In parallel programming, however, the instructions are executed in parallel on multiple cores/processors of a parallel system. Multiple independent processes take place at the same time by executing multiple threads on different cores. The three well known repetitive constructs for implementing parallelism in, C# language, are as follows:

1. **Parallel For Loop:** is the basic parallel construct that improves performance by utilizing the multi-core architectures. The improvement is proportional to the number

of cores being used for executing a given task (Campbell et al., 2010). The threads are managed automatically and they may not follow the exact order of execution as in sequential approach. The first two arguments, in parallel for, provide the lower and upper bounds while the third is an action that is invoked once per iteration (Loidl, 2015).

2. **Range partitioning:** is applied to those data structures that are based on indices such as arrays and where the length is known in advance. To process a specified range, each thread is provided with unique indices for the beginning and ending of the given range.

   It explicitly makes partitions and reduces the overhead of parallelism for an improved performance (Loidl, 2015). It offers good performance and allows to design our own partitioning strategy that suits our needs or the data source (Campbell et al., 2010).
3. **Chunk Partitioning:** is used with data sources of unknown length such as link lists. In chunk partitioning, each thread gets some number of elements in a chunk, processes them and come back to get additional elements. It ensures the distribution over all cores and avoids duplicate assignments. Chunk partitioning balances the load among cores and each core dynamically request more work as needed after the completion of already assigned tasks (Campbell et al., 2010; Loidl, 2015).

THE PROPOSED PARALLEL ALGORITHM

This section introduces the proposed parallel LOAR algorithm in an abstract form and, then, evaluates it for two different implementations. It provides the reduced impact of the proposed method on the re-allocation process of regions.

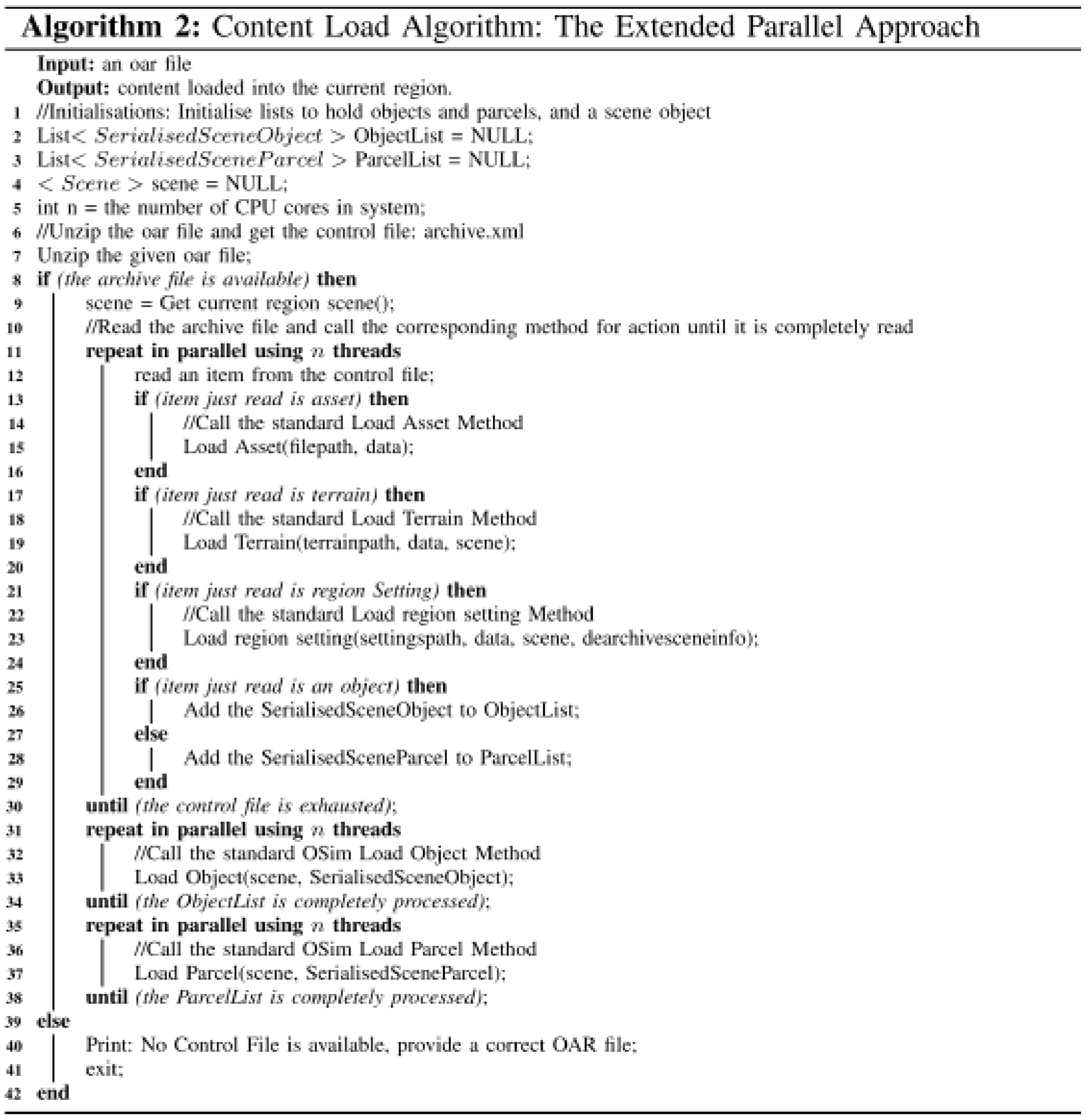
1. The Proposed Parallel LOAR Algorithm

Algorithm 2 extends the sequential algorithm presented in Algorithm 1 to use it over multi-core architectures. The parallel algorithm determines the number of processing cores available in the system hosting the region. The parallelism is, then, applied not only to the repetitive construct in line 10 of Algorithm 1 to parallelize reading the control file and processing the items accordingly but also looping constructs in line 30 and 34 for loading the objects and parcels to a region in corresponding order. These changes are reflected in line 11, 31 and 35 of Algorithm 2. The parallel algorithm acts the same way as sequential algorithm but processing various activities in parallel using different processing elements.

2. Evaluation and Results

The main goal of this work was to determine the impact of using the existing parallel multi-core architectures for Load OAR (LOAR) algorithm of the OAR functionality. Table 1 describes five different example worlds content, used in this work for evaluation purposes, in terms of the number of primitives, assets, objects, and scripts. Though, a region could have more than one parcels. Each of the example worlds content, used in this work, comprises of a single parcel as each of them provides content for a single region owned and managed by a single user.

This work uses the same experimental set-up presented in INVESTIGATING THE SEQUENTIAL ALGORITHM section for investigation purposes. In start of each experiment, we create an empty region which is, then, loaded with one of the example world’s content shown in Table 1 using the current and proposed methods. Loading times for each instance is recorded which are used for comparison purposes.



Algorithm 2: Content Load Algorithm: The Extended Parallel Approach

```
Input: an oar file
Output: content loaded into the current region.
//Initialisations: Initialise lists to hold objects and parcels, and a scene object
List< $SerialisedSceneObject$ > ObjectList = NULL;
List< $SerialisedSceneParcel$ > ParcelList = NULL;
< $Scene$ > scene = NULL;
int n = the number of CPU cores in system;
//Unzip the oar file and get the control file: archive.xml
Unzip the given oar file;
if (the archive file is available) then
    scene = Get current region scene();
    //Read the archive file and call the corresponding method for action until it is completely read
    repeat in parallel using $n$ threads
        read an item from the control file;
        if (item just read is asset) then
            //Call the standard Load Asset Method
            Load Asset(filepath, data);
        end
        if (item just read is terrain) then
            //Call the standard Load Terrain Method
            Load Terrain(terrainpath, data, scene);
        end
        if (item just read is region Setting) then
            //Call the standard Load region setting Method
            Load region setting(settingspath, data, scene, dearchivesceneinfo);
        end
        if (item just read is an object) then
            Add the SerialisedSceneObject to ObjectList;
        else
            Add the SerialisedSceneParcel to ParcelList;
        end
    until (the control file is exhausted);
    repeat in parallel using $n$ threads
        //Call the standard OSim Load Object Method
        Load Object(scene, SerialisedSceneObject);
    until (the ObjectList is completely processed);
    repeat in parallel using $n$ threads
        //Call the standard OSim Load Parcel Method
        Load Parcel(scene, SerialisedSceneParcel);
    until (the ParcelList is completely processed);
else
    Print: No Control File is available, provide a correct OAR file;
    exit;
end
```

a. Implementations

The proposed work extends the OSim functionality and, therefore, C# language is used to develop both the proposed parallel algorithm for content load mechanism. Two variations of the parallel algorithm are developed: one using the basic parallel for construct while the other using chunk partitioning. Time factor is used for comparison purposes in this work.

b. Results and Discussion

Table 3 summarizes the results of experiments for both the standard and proposed algorithms for loading different VWs content to the regions. This table includes the improvement by the parallel implementations in percent (%). Both the improvements provide reductions directly against the timings recorded for the sequential algorithm.

Table 3 The comparison of sequential and parallel content load algorithms.

Example World Content (Char)	Timing Information			Time Improvement by Parallel Approaches	
	Sequential Approach (Sec)	Parallel Approaches			
		Parallel For (Sec)	Chunk Partitioning (Sec)	Parallel For (%)	Chunk Partitioning (%)
IST OAR (ARL-STTC, 2020)	180	87	62	48	66
Alfea OAR (Artis, 2020)	250	123	71	51	72
Epic-Citadel (Cuteulala, 2020)	177	83	56	53	68
OSim-openvce (Edin.Uni, 2020)	59	31	22	47	63
GoneCity (Haan, 2020)	65	28	21	57	68

It shows that parallel for obtained a clear improvement over the sequential approach. The parallel implementation that used chunk partitioning, however, obtained an optimal result. In general, the parallel for, got a minimum of 47% and a maximum of 57% improvement. The minimum gain is based on actual timings of 59 and 31 seconds for sequential and parallel implementations against loading the OSim-openvce content. The maximum value is based on actual timings of 65 and 28 seconds for sequential and parallel implementations against the content named GoneCity. The chunk partitioning, however, had a minimum of 63% and a maximum of 72% improvement. The minimum gain is based on actual timings of 59 and 22 seconds for sequential and parallel implementations against the OSim-openvce content. The maximum value is based on actual timings of 250 and 71 seconds for sequential and parallel implementations against the content called: Alfea. The rest of the results are self-explanatory and are, therefore, not further explained.

Table 4 Regional transfer timings based on sequential and parallel content load mechanisms.

	Timing Information		Time Improvement by Parallel Approaches
	Sequential	Parallel Approaches	

Example World Content (Char)	**Approach (Sec)**	**Parallel For (Sec)**	**Chunk Partitioning (Sec)**	**Parallel For (%)**	**Chunk Partitioning (%)**
IST OAR (ARL-STTC, 2020)	196	103	78	47	60
Alfea OAR (Artis, 2020)	270	143	91	47	66
Epic-Citadel (Cuteulala, 2020)	192	98	71	49	63
OSim-openvce (Edin.Uni, 2020)	71	43	34	39	52
GoneCity (Haan, 2020)	73	36	29	51	55

This work used a very restricted standalone environment based on SQLite database, the responses could be further improved when more stable and scalable grid environment is used. This work used a newly created region in each experience, the use of existing regions might take more time when the content is loaded to them as they delete the existing content before loading the new content.

3. The Impact of Parallelism on Re-allocation Process

This section explores the impact of parallel content load algorithm on regional transfers. Both implementations of the proposed algorithm are applied to the 5 example worlds content presented in Table 1 along with responses for the sequential algorithm when applied to them. In fact, the results presented in Table 4 extends the results calculated and presented before in Table 2 for determining the impact of sequential algorithm on the re-allocation process of a region.

Table 4 not only shows the timings taken by the re-allocation process for sequential and parallel approaches but also the improvements in percent (%) by both the parallel implementations. The improvements are both with respect to the responses recorded for sequential approach. It is clear from the results that when a region having IST content was transferred, the parallel for implementation, reduced the timings by 47% compared with the time recorded for sequential algorithm. Chunk partitioning reduced it by 60%.

The parallel for and chunk partitioning methods for Alfea, Epic-Citadel, OSim-openvce and GoneCity reduced the timings by 47%-66%, 49%-63%, 39%-52% and 51%-55% against the timings taken by the sequential algorithm, in given order. In short, it is learnt that parallel for has a minimum of 39% while a maximum of 51% improvement. However, chunk partitioning obtained a minimum of 52% while a maximum of 66% improvement.

CONCLUSION

This paper investigated the current content load algorithm of OSim framework which is one of the major components of the scalability framework presented in Farooq & Glauert (2017b). It identified the huge impact of this algorithm on the re-allocation process of a region and, then, explored the use of parallelism of multi-core architectures to encounter this impact. A parallel content load algorithm was proposed as an extension to the current algorithm in such as way the structure of the proposed algorithm resembles the existing algorithm. It only replaced the repetitive sequential constructs with the parallel constructs of C# language. The space-time complexity of the proposed algorithm, therefore, remains the same. Similarly, it places no extra burden in terms of cost, as it uses multiple processing elements of the current systems. This algorithm was implemented using two well-known parallel programming constructs (the parallel for and chunk partitioning) of C# language.

Simulation results showed that both the implementations of the proposed parallel algorithm minimized the time taken by content load involved in a transfer. The parallel for implementation with the help of four cores processor processes four different items in parallel during an iteration, it performed better than the sequential approach. This approach, however, due to using a conservative locking mechanism, introduced longer delays in the iterations having time consuming items to process. In all such cases, the cores with smaller tasks must wait for the cores assigned bigger tasks to finish before starting the next pass. The implementation using chunk partitioning showed an optimal behaviour for the current scenarios, as it overcome the inherent issue of basic parallel for implementation. Considerable improvements were observed when both the implementations of the proposed mechanism were used in regional transfers instead of sequential algorithm for loading the content.

FUTURE DIRECTIONS

The following are listed as future directions based on limitations of the proposed work and its current implementations:

1. The proposed algorithm was extended by replacing the sequential constructs with parallel ones. However, we believe that developing a parallel alternative from scratch would obtain further improvements over the current parallel algorithm.
2. It would be interesting to evaluate the proposed mechanism with the help of more advanced, stable and efficient environments such as the one based on MySQL or MSSQL.
3. It would be interesting to test the extended algorithm on grid and cloud infrastructures.
4. It would be interested to investigate the impact of the proposed algorithm using an increased number of processing elements.
5. This work could be explored for more detailed content in future. It would be interested to develop time prediction models for both sequential and parallel algorithms.
6. It would be interesting to test the proposed algorithm on GPU based systems and compare the responses with CPU based systems calculated in this work.

REFERENCES

Abdulazeez, S. A., & Rhalibi, A. 2018. Dynamic load balancing for massively multiplayer online games using OPNET," in International Conference on E-Learning and Games. Springer, pp. 177–191.

Alahuhta, P., Nordbck, E., Sivunen, A., & Surakka, T. 2014. Fostering team creativity in virtual worlds. Journal for Virtual Worlds Research, 7(3).

ARL-STTC. 2020. IST-OpenSim Archive (OAR). http://www.outworldz.com/Sculpts/cgi /files/IST%20OAR/. (Accessed: March, 2020)

Artis, C. 2020. Alfea 3 - OpenSim Archive (OAR). http://www.outworldz.com/Sculpts/cgi /files/alfea3%20OAR/. (Accessed: March, 2020)

Ata, R., & Orhan, S. 2013. An implementation of virtual worlds platform for educators in second life. Procedia - Social and Behavioral Sciences, 2nd World Conference on Educational Technology Research.

Behnke, L. 2020. Increasing the Supported Number of Participants in Distributed Virtual Environments, Ph.D. dissertation, School of Computing Sciences - University of the West of Scotland, West Scotland, UK.

Bredl, K., Gro, A., Hnniger, J., & Mller, J. 2012. The avatar as a knowledge worker? How immersive 3D virtual environments may foster knowledge acquisition.

Campbell, C., Johnson, R., Miller, A., & Toub, S. 2010. Parallel Programming with Microsoft .NET: Design Patterns for Decomposition and Coordination on Multicore Architectures. Microsoft Press, USA, 1st edition, 2010. ISBN 0735651590.

Cuteulala, A. 2020. Epic-Citadel OpenSim Archive (OAR). http://www.outworldz.com/ Sculpts/cgi/files/Epic-Citadel-OAR/, Accessed: January, 2020.

Edin.Uni. 2020. Open Virtual Collaboration Environment (OpenVCE.net). http://openvce.net (Accessed: March, 2020)

Elfizar, Baba, M. S., & Herawan, T. 2019. “1P10: A large scale distributed virtual environment,” in Proceedings of the International Conference on Data Engineering (DaEng-2015), pp. 21–29.

Farooq, U., & Glauert, J. 2017a. Faster dynamic spatial partitioning in OpenSimulator. *Virtual Reality*, 21(4):193–202.

Farooq, U., & Glauert, J. 2017b. Integrating dynamic scalability into the OpenSimulator framework, Simulation Modelling Practice and Theory, vol. 72, pp. 118–130.

Farooq, U., Rabbi, I., Zia, K., Israr, S. Z., & Shereen M. 2018. Quicker re-allocation of regions in OpenSimulator framework, in Proceedings of the 2nd International Conference on Future Networks and Distributed Systems, ICFNDS’18. New York, NY, USA: ACM, pp. 3:1–3:7. Available: http://doi.acm.org/10.1145/3231053.3231056

Haan, R. 2020. Gone City OpenSim Archive (OAR). http://www.outworldz.com/Sculpts/cgi /files/Gone%20City%20OAR/. (Accessed: Jan, 2020)

Hakonen, M., & Bosch, S. P. 2014. Virtual worlds enabling distributed collaboration. Journal for Virtual Worlds Research, 7(3).

Harris, D. M., & Harris, S. L. 2013. Digital design and computer architecture. Elseveir Inc.

IDC. 2004. Butterfly.net: Powering next Generation Gaming with On-Demand Computing (Tech. Rep.). IBM: An IDC e-Business Case Study.

Kim, J. H. 2020 A hybrid cloud-p2p architecture for scalable massively multiplayer online games, *The Journal of The Institute of Internet, Broadcasting and Communication,* vol. 20, no. 3, pp. 73–81.

Linden Research, Inc. 2021. Second Life: The Official Website, https://secondlife.com/, last accessed: April, 2021.

Liu, H., & Bowman, M. 2010. 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 (pp. 43–52). Washington, DC, USA.

Loidl, H. W. 2015. Parallel Programming in C#. School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh. (Lecture Notes)

OpenSim. 2021. OPENSIM-EDU: Resources for Educators using OpenSim. http://opensim-edu.org/blog/. (Accessed: July, 2021)

Osgrid.org Inc. 2021. OSGRID - The Open Source Metaverse, http://www.osgrid.org, (last accessed: June, 2021).

OSim-Wiki. 2021. OpenSimulator: An introduction, http://opensimulator.org/wiki/Main Page, (last accessed: June, 2021).

Prendinger, H., Jain, R., Imbert, T., Oliveira, J., Li, R., & Madruga, M. 2018. Evaluation of 2D and 3D interest management techniques in the distributed virtual environment DiVE, *Virtual Reality*, vol. 22, no. 3, pp. 263–280.

Rogers, T. 2021. Should you host on the Amazon cloud? http://www.hypergridbusiness.com/2013/11/should-you-host-on-the-amazon-cloud/, last accessed: April, 2021.

Shahvarani, A., & Jacobsen, H. A. 2019. Parallel index-based stream join on a multicore CPU.

Sox, C. B., Crews, T. B., & Kline, S. F. 2014. Including virtual and hybrid meeting planning within the curriculum: A knowledge management perspective. *Journal of Hospitality & Tourism Education*, 26(3), 147–152, doi: 10.1080/10963758.2014.936258.

Spieldenner, T., Byelozyorov, S., Guldner, M., & Slusallek, P. 2018. FiVES: an aspect-oriented approach for shared virtual environments in the web. *The Visual Computer* **34,** 1269–1282. https://doi.org/10.1007/s00371-018-1564-0

Yilmaz, L., Taylor, S., Fujimoto, R., & Darema, F. 2014. Panel: The future of research in modeling and simulation. In Simulation Conference (WSC), 2014 Winter, pages 2797–2811, Dec 2014. doi: 10.1109/WSC.2014.7020122.