Highly Scalable Server Architecture for Massive Multi-player 3D Virtual Spaces

MOLDOVEANU ALIN DRAGOS BOGDAN, MOLDOVEANU FLORICA, ASAVEI VICTOR
The Faculty of Automatic Control and Computers
University “POLITEHNICA” of Bucharest
Splaiul Independentei 313, Bucuresti, Cod Postal 060042
ROMANIA
alin.moldoveanu@rdslink.ro, fm@cs.pub.ro, victor.asavei@cs.pub.ro http://cs.pub.ro

Abstract: - 3D massive multiplayer virtual spaces are getting more and more popular, not only as computer games but as complex simulation and interaction environments, heading to become the next paradigm of multi-user interface. Still their universal adoption is hindered by some serious practical issues, mainly revolving around development costs and scalability limitations. The authors consider that the main cause for these limitations resides in the particularities of server-side software architectures - traditionally designed as clusters of single processor machines. The paper gives a brief overview of current solutions and their limitations and propose two innovative architectural concepts which have a big potential for creating cheaper and more scalable solutions. We describe a region based decomposition of the virtual space together with supporting middlewares of messaging, distributed control and persistence, which allow an efficient and flexible work effort distribution on server side. The solution allows for both horizontal and vertical scalability The vertical scalability is then mapped in an innovative manner on the last generation of SIMD-like multi-processor graphics cards. The huge processing power of these cards, with the right architecture, can take over the bulk of the server-side effort. Our prototype tests indicated that the solution is feasible and may represent an important turnaround in the development of more scalable and much cheaper massive multi-player server architectures for various types of virtual spaces.

Key-Words: - 3D, virtual space, massive multiplayer, server scalability, GPGPU, CUDA

1 Introduction

3D massive multiplayer virtual spaces offer rich information delivery, interaction and collaboration possibilities. They are getting more and more popular each day, not only as computer games but also as simulations, training, edutainment or work-oriented applications. Considering the huge attraction and interest from the users, it is highly probable that not far in the future they will be the standard user-interface paradigm.

Right now, the market adoption of such virtual spaces is hindered by the huge costs involved in their creation, operation and maintainance. Creation costs regard software development and content creation. Content creation is basically open to everyone, with many easy to use tools available, but the software creation is prohibitive.

The architecture for such applications is typically client-server, with the server managing the content and connectivity to the virtual world. Particularities of the domain are the complexity of interactions, the multi-player aspect, requirements about persistence and up-time, and, most important, the short response time (round trip latency) that users expect. All of this, put together, will make the development of a 3D multi-player virtual space available only to a selected elite of software developers and, of course, very costly. Operation and maintainance cost tend to be very high due to the traditional approach to server design for this type of application.They provide decent scalability, but, at some point, they get limited in this aspect.

The paper proposes a high level architectural solution designed for high scalability, both horizontal (on multiple computers) and vertical (on multi-processor machines). Then we show how the vertical scalability can be mapped to benefit from the huge parallel power of the new generations of graphics processing units.

2 Server Responsibilities and Issues

2.1 Server responsibilities

The vast majority of 3D massive multiplayer virtual spaces are implemented as client-server applications. This comes from the competitive and security issues existing in most of them. Even if, technically, some virtual space might have more efficient peer to peer solutions, this is excluded because is possible, actually quite probable, that some users will exploit the inherent weakness of the architecture by reverse engineering and modifying the client.

Hence, the first role of the server, which force the client-server approach, if to be the unique supervisor and arbiter of the virtual space.

To accomplish this role, the server must:

- keep a full semantical representation of the internal state of the virtual space
- validate the significant actions of each user
- detect patterns of illegal actions and possible frauds

Besides this arbiter role, typical responsibilities of the server are:

- characters/avatars creation and changing
- login system
- implementing the non-user controlled aspects of the virtual space logics:
  - events
  - NPCs (non-player characters)
  - mobs control
  - quests
  - artificial intelligence
- permanent updating each client about all the events of interest for him

2.2 Scalability issues for the server

As we will argument below, the scalability of a massive multiplayer virtual space server is substantially different from other classical scalability problems.

For example, for applications consisting of highly intensive more or less complex database accesses and queries, scalability is almost entirely solved by classical techniques at database or file system level. There are no synchronization problems that can't be solved with the basic locking of records or tables. There are no real time challenges. The clients for these applications also have usually small complexity, they just acting at presentation level - GUI.

Other applications, like search engines, rely on specific algorithms for dividing a query in several smaller ones, distributed over several machines .

Most of these applications do not have to address the issues typical for a massive multiplayer virtual space:

- permanent connections to a large number of clients
- high amount of information exchanged in real time (users actions and their effects)
- the absolute demand for a fast, guaranteed response time to users actions (round-trip latency)
- the relative complexity of the logics and interactions in the virtual space, which usually implies:
  - there is no straightforward full hierarchical tree decomposition of the problem
  - even if a decomposition is found bases on some criteria, the resulting subtasks will require some amount of communication between them

2.2.1 Low latency

Virtual spaces are mostly used in entertainment (games). From other uses, they also make up good simulation and training environments. In both cases, the users do expect and need the response to their actions and to other users actions in real time, as fast as possible. A usual value is in the rank of tens to hundreds of milliseconds, anything above affecting the quality of the interaction (is usually called lag and hated by the users).

This is so important that some specific techniques like "guessing" the results of an action or the expected next position during movement, followed by later corrections or adjustments when necessary, are sometimes employed to minimize latency, as observed by the user.

Such techniques are anyway more or less just workarounds or tricks. Above them, is essential the architectural design of the whole system to minimize the latency.

2.2.2 Strongly linked multi-user interactions

Unlike other application types, where the work flow is initialized and directed by a single user, virtual spaces support complex interactions, involving many users at the same time. The actions of an individual user and their effects, usually calculated by the server, must be propagated in real time to all those affected by them..

Obviously this requires to a communication architecture with support for broadcast. This has no impact over scalability

However, of maximum importance for scalability is the accomplishment of the following thumb-rule:

- the division and distribution of users and tasks over different processing units (processor or computers) must allow very fast retrieval of the necessary information about all the users potentially affected by an action

This

- designing the scenario of the virtual space, the geography of the world, the possible interactions between users and the quests
- the internal data structures on storage
- the server communication middleware

3 Limitations of the Current Solutions

Due to legacy background and education, most massive multiplayer virtual spaces developers are oriented to classical architectures, with a very small number of precesses or thereads, since the time of single-processor computers. Hence, first scallability attempts followed this approach, ending to me more or less like some partial work-arounds for the problem.

3.1 Virtual world independent instances

Totally independent instances of the virtual world, usually called realms in games terminology, are created and reside on completely independent servers. Users can acces any of this instances, but, as any moment in time, the interraction is limited server-wide and generally, character/avatar transfer between worlds is restricted or limited. Sometimes, minimal connection between these world exists though, but they are minimalistic and don't have real time behaviour.

3.2 Instance dungeons

A similar workaround is the instance dungeon concept – a subspace of the virtual space, for which an independed copy is spawned each time an user or a group of users is accessing it. Users in a dungeon can only interract with others in the same dungeon.

From users point of view, this limits the competion for resources in that area to an acceptable degree.

However, from developers point of view, is a tricky way to:

- reduce server workload by limiting the number of users in an dungeon and therefore reducing the overall number of interractions
- divide the server effort, since the dungeons are quasi-independent and they can be hosted on different computers.

3.3 Statical spatial decomposition

Sometimes, the geography of the world can be exploited or, even more, can be designed to be decomposed in several totally-independent regions. For example a virtual world made of planets, or islands, or having some non-passable natural barriers can benefit from this idea. The transition from one region to another is done through some key points and usually is not instantaneous.

Hence, the server workload can naturally be divided by these regions, each being handled by a different processing unit, usually a different computer.

This approach is a nice workaround for many cases, still it has some serious limitations or disadvantages:

- the resulting virtual world lacks total spatial continuity
- some computers corresponding to overcrowded regions may not be able to while ,resulting in lag ,handle all the workload for empty or scarcely populated ,others are not used at their capacity ,regions
- the processes that ,to obtain realtime speed handle each region are usually defined and ;statically allocated on different computers transferring a region process from one omputer to another might be costly in terms c -of runtime speed and is usually avoided limitation to good load balancing

Is obvious that this solution is still a workaround to the real scalability issue.

4 Dynamical Spatial Decomposition

We suggest a more powerful approach, that would divide the virtual space into regions dynamically, at runtime.

4.1 The decomposition method

It is based on some heuristic function that calculates the workload for a region. When the function value exceeds a threshold, the region can be subdivided into smaller regions, and so on.

The heuristic can take into account:

- number of users in the region
- number of mobs in the region
- number of inanimate objects in region
- dynamics of interactions, etc.

The dynamic decomposition will have a tree-like structure. A binary or quad-tree fits perfect this idea. The decomposing process should stop when each region computes a value for the heuristic lower then the threshold, or the region dimensions are lower then a given constant, usually depending on the users and mobs perceiving range.
This decomposition allows:

- reduction of the workload, as each users actions will be tested/performed only on users in it's region (or neighboring, if user is on close to border)
- division of the workload, each region or groups of regions being assigned to different processing unit

The mechanism is exemplified in Fig. 1: the dots represent the users, the heuristic is very simple, equal to the number of users withing the region, and the decomposing threshold is 4. Basically, a region will be divided when noUsers(region) >= 4. Quad-tree is used. At the moment of the last division, it is also decided that the processing unit 1 cannot handle all the workload, therefore some of the resulting regions will be assigned to processing unit 2.
The division in regions, and also the assignment of regions to processing units can be highly dynamic.

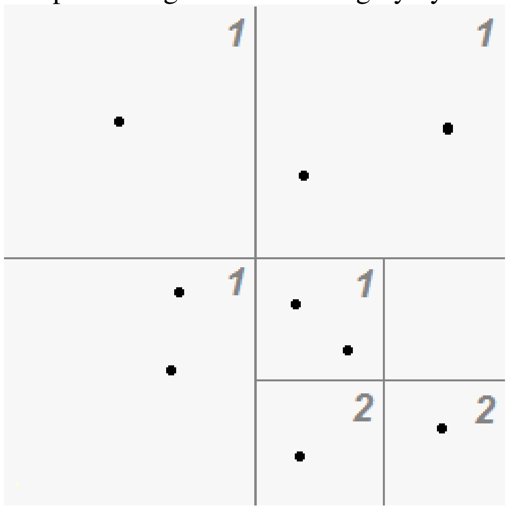

Fig. 1

When the heuristic workload estimation value for a previously divided region goes back under the threshold, its subregions can be grouped back into the original region to reduce the tree.

4.2 Required Middlewares

To support this highly dynamic division and load balancing possibilities, some underlying middlewares are required. They must allow:

- fast transfer of users between the regions processing tasks that run on different computers
- fast transfer of a region data from one computer to another
- Scalability

As this paper focus mostly on the decomposition and on scalability issues, we will just give a brief overview of the needed middlewares and our solutions for each of them:

4.2.1 Control layer

The conceptual model of a virtual space is highly event based - basically all changes in system state are caused by users actions.
Therefore, the control layer will be responsible only with initializing the system and supervising the regional division and eventually load ballancing.
Any other tasks that are not handled directly by the regions (like commerce, chat, etc.) should be well separated from the main control layer, for example using a plugin system.

4.2.2 Communication layer

Considering the nature of the system and the scalability requirements, we consider that the best choice will be a publish-subscribe messaging system.
This is basically a paradigm of communication through asynchronous messages where transmitters (also names publishers or producers) do not sent the messages to a specific destination. Rather, the messages are classified in classes and have attributes. The subscribers declare their interest for some classes of messages or having some attributes and will receive only the corresponding messages.
Main advantage of this method is the total decoupling between producers and consumers, this usually allowing for great scalability
There are many variants and implementation topologies for this paradigm. Without going into many details, we have decided that best choice is:

- message filtering based on type and content, with the possibility of specifying very strong filters
- domain server topology
- low level networking service tuned to transmit very fast the small sized messages, these being the huge majority of the messages in our case

With publish-subscribe, moving a user from a region to another comes to changing subscriptions, updating the domain server and sending a message with essential real time user data.

4.2.3 Persistence layer

Traditionally, this layer is used just to save periodically esential system state, more like a backup system.
However, recent advances in database engines, combined with inteligent caching, allow its extended use to store real time system data that is used by the regions, thus allowing easy transfer of the regions tasks from one computer to another.

5 Vertical scaling using GPGPU

Previous section has described the dynamical regional decomposition and middleware components required for horizontal scalability
In this section we describe a brand new idea that builds on the decomposition idea to make use of the huge parallel computing power of the new generations of graphics processing units.

5.1 GPUs evolution

A graphics processing unit (GPU) is a specialized hardware module, used as component of a PC, graphics station or gaming console, specialized in performing high speed graphics calculations.
GPUs spectacular evolution can be briefly resumed:

- First GPUs only supported basic graphics functions; as hardware they were adapted general purpose co-processors or signal processors
- PC market boom turned the GPU into a standard component, taking over most standard 2D and 3D graphics calculation
- Programmable shaders were introduced, beeing the first form of GPU programmability
- General-Purpose computation on GPUs (GPGPU) is the technique of using a GPU to perform computation in many applications traditionally handled by the CPU – made possible by the addition of programmable stages to the rendering pipelines and various libraries and development toolkits
- In the near future the trend of turning dedicated GPUs architectures into more general parallel ones will certainly continue, together with some hybrid solutions, like announced Intel's Larrabee

5.2 Current GPGPU concepts and limitations

5.2.1 Streams

The closest concept to the hardware and programming model of a modern GPU is that of a SIMD machine, or stream processing.
Streams are sets of records with identical format that require simillar processings. In the case of GPGPU, the most natural format for a stream is that of a 2D grid, which fits with the rendering GPU model.

5.2.2 Kernels

The processings or functions that work on stream are usually named "kernels".
For example, vertex or fragment shaders are particular cases of kernels.
In the particular case of GPUs, the kernels can be seen as the body of loops iterated over 2D matrixes.

5.2.3 Flow control and limitations

Usual if-then-else or loops mechanism have been only recently added to GPU. Some limitations, total or partial still exists, for example:

- runtime ramifications come with a big performance cost
- recursivity is not supported and can just be partially emulated
- data transfer from/to memory is also costly in terms of performance

We must highlight that a runtime branching followed by a barier synchronization for a group of threads running same kernel comes at a cost of hundreds of normal operations. Same for data transfer to/from memory.
This makes the GPGPU only suitable for some kinds of operation, that require few data transfers and have a high degree of parallelism.

5.1 Regional decomposition with GPGPU

The challange is to distribute in an eficient way the workload between CPU si GPU.
The following aspects must be considerred:

- Granularity of kernels must be small and their nature highly parallel
- The coordination performed by the CPU and the data transfer CPU-GPU must be minimized
- Workload distribution over streaming processors must be adaptable in real time

5.3.1 GPU tasks

GPU should only handle computational intensive tasks, organized in low granularity kernels:

- collision detection

- advanced phisics calculations
- basic decisional AI

The code executed by GPU will have two components:

- a component that creates the kernels, assign them to processor groups and launches them
- the kernels

5.3.2 CPU tasks
Will be responsible for the high level control:

- regions decomposition
- input/output
- distributed control, messaging, persistence
- sinchronization with GPU

5.3.3 Frames, Execution and Synchronization
The server workflow consists of frames. A frame is a very short time inteval, which, as a design choice, can be fixed or variable.
During a frame the following things will happen:

- user input is take from input queues, pre-processed and passed to the GPU code of control.
- GPU will interpret input and create kernels to handle it. For example, a colision detection kernel will activate for each pair of moving objects
- kernels are launched over streaming processors
- the streaming processors execute the kernels
- when all kernelurile are completed (variable frame design) or the frame duration elapsed (fixed frame) the available results are placed in output queues

5.3.4 Load balancing
A computer can handle some number of regions. Each region is assigned a number of GPU streaming processors, according to its computational needs.
As this number can be changed dynamically, the load ballancing comes naturally at no cost.

5.4 Prototype and results
The solution described above was tested by creating a server prototype, during the “Graphics and Virtual Reality Workshop 2008”, which took place in summer, in the University “POLITEHNICA” from Bucharest.
The arhitectural concept was implemented using the CUDA and NVIDIA graphics cards.
The prototype only included basic elements for testing:

- basic TCP/IP multi-player communication
- CPU and GPU implementation of the regions decomposition and management concept
- GPGPU colision-detection
- basic client to tun the tests
- tests creating collisions between huge number of objects in the 3D virtual space

The results were really encouraging and we are determined to further explore the possibilities of the concept.

6 Conclusion
If the proposed architectural solution will be successfuly implemented, we can see, in the near future that the hundreds of computers server farms for MMORPGs being replaced with only a few Pcs equiped with multiprocessors GPUs.
The GPGPU is rapidly evolving and the new hybrid solutions will certainly affect server architectures, massive multiplayer virtual spaces beeing one that will mostly benefit of it.
Full spatial continuity and scalability for a virtual space, cheap and flexible hosting and maintainance will contribute to their turnout as the next user-interface paradigm.

References:

[1] J. Waldo, Scaling in Games and Virtual Worlds. *Communications of the ACM*, Vol 51, No 08, 2008, pp 38-44.

[2] Ta Nguyen, Binh Duong, Suiping Zhou, A dynamic load sharing algorithm for massively multiplayer online games, *The 11th IEEE International Conference on Computational Science and Engineering*, 2003, pp 131-136.

[3] P. Morillo, J. Orduña, M. Fernández, Workload Characterization in Multiplayer Online Games, *ICCSA (1)*, 2006, pp 490-499.

[4] S. Ferretti, Interactivity Maintenance for Event Synchronization in Massive Multiplayer Online Games, *Technical Report UBLCS-2005-05*, March 2005.

[5] S. Bogojevic, M. Kazemzadeh, *The architecture of massive multiplayer online games*, M.S. thesis, Lund Institute of Technology, Lund University, Lund, Sweden, 2003.

[6] *Multiverse Platform Architecture*, http://update.multiverse.net/wiki/index.php

[7] *ActiveWorlds*, http://www.activeworlds.com/