Analyzing the Performance of a Cluster-Based Architecture for Immersive Visualization Systems

P. Morillo [a], A. Bierbaum [b], P. Hartling [b], M. Fernández [a], C. Cruz-Neira [c]

[a] *Instituto de Robótica. Universidad de Valencia*
Poligono de la Coma, S/N
46980 Paterna (Valencia) - Spain
Phone Number: +34-963543673
Fax Number: +34-963543550
Contact Address: pedro.morillo@uv.es

[b] *Infiscape*
2901 South Loop Drive
50010 Ames (Iowa) - USA
Phone Number: +1-5152963787
Fax Number: +1-5152969910
Contact Address: aronb@infiscape.com

[c] *LITE. University of Louisiana at Lafayette*
537 Cajundome Boulevard
70506 Lafayette (Louisiana) - USA
Phone Number: +1-3377355483
Fax Number: +1-3377351346
Contact Address: carolina@lite3d.com

Abstract

Cluster computing has become an essential issue for designing immersive visualization systems. This paradigm employs scalable clusters of commodity computers with much lower costs than would be possible with the high-end, shared memory computers that have been traditionally used for virtual reality purposes. This change in the design of virtual reality systems has caused some development environments oriented towards shared memory computing to require modifications to their internal architectures in order to support cluster computing. This is the case of VR Juggler, which is considered one of the most important virtual reality application development frameworks based on open source code.

This paper not only describes in detail the mechanisms based on cluster computing included in the internal design of VR Juggler, but also proposes a new global performance evaluation methodology. The goal of this methodology is to test the

graphical performance of immersive visualization systems based on clusters of computers in terms of both network latency and number of nodes in the cluster. In this sense, a performance evaluation of VR Juggler, both in an overall and a modular approach, is presented. The obtained results show that VR Juggler can be considered as an efficient tool to support immersive visualization systems on a cluster of computers.

Key words: Cluster computing, Performance Evaluation, Immersive Visualization, Distributed Systems

1 Introduction

Immersive visualization environments are virtual reality (VR) systems where users can view, navigate and/or modify three dimensional models with a first-person perspective. The type of immersion achieved in these graphical systems let users behave in the 3D virtual scene in the same way they would behave in a real environment. Immersive visualization (IV) systems are currently used in different applications such as scientific data visualization (metabolic networks [13], fluid dynamics [40], information flows [25]), e-learning [17], collaborative design [15,27] or computer games [24].

Traditionally, IV systems have been designed on dedicated, high-end, shared memory computers to generate interactive virtual environments. These systems typically handle multiple graphics outputs (monitors, projectors, flat panels, etc) at the same time, and this type of multi-screen system must not be confused with distributed virtual environment (DVE) systems, where many users remotely connected from different computers (typically connected through the Internet) share the same 3D virtual scene [36]. Depending on how the set of screens is organized, a wide variety of IV devices offer different sense of presence, presence and interaction capabilities in the 3D virtual environment. Examples of IV devices are CAVEs [12], VR workbenches, flat-screen walls, curved-screen theaters and concave-screen domes [41]. IV systems usually require one or two video outputs for each projection surface, and they often use many input devices simultaneously. In recent years, the almost exclusive use of high-end computers for these purposes has shifted to commodity hardware, since it has become a low-cost alternative [2,23,37]. Continuous, rapid improvements in commodity hardware have allowed designers of immersive visualization systems to employ high-quality graphics hardware, high-speed processors, and significant amounts of memory with much lower costs than

⋆ Supported by the Spanish MEC under grants CSD2006-00046, TIN2006-15516-C04-04 and PR2006-0526

would be possible with high-end, shared memory computers. However, in order to handle multi-screen IV systems using a set of commodity computers it is necessary to add a middleware layer to the system. This software layer, included within the software suite for the development and execution of IV applications, must be in charge of mimicking the behavior of a single, shared memory computer and showing the system to the application developer as a single unit. This transparency of the VR system can be accomplished through the use of a tightly synchronized cluster.

Figure 1 shows the schematic diagram of a typical IV system based on a cluster of commodity computers. These systems are also called *graphics clusters* [23,33]. In this case, the system is composed of three computers (connected through a Fast Ethernet switch) where each of them contributes to compose a section of the final 3D virtual scene. This figure shows how outputs of the three graphics cards are configured to be horizontally concatenated providing a side-by-side display. Since each single computer of the cluster has a maximum graphics resolution (for instance 1024x768 pixels), they only draw a section of the scene and their graphics outputs are synchronized in order to recreate a virtual monitor with an improved resolution. This feature is also shown in Figure 1 where the high-definition representation of a Romanic Temple is achieved in a 3072x768 seamless virtual monitor by means of three computers executing ClusterJuggler. Since most of the development tools of IV systems are multi-platform, the cluster nodes can be equipped with different operating systems, or they can even be designed for different hardware architectures. These type of systems are called heterogeneous clusters [7,32,37].

Although graphics clusters seem a typical example of tightly synchronized clusters, these systems are not always composed of a set of computers interconnected by a high-performance switched network (Gigabit Ethernet, Infiniband, Myrinet, etc [6]). Not all the computers of the cluster are directly connected to the inputs of the visualization device in many configurations of IV systems. The reason for this design is because many 3D virtual environments contain physical models representing exactly the behavior of complex elements of the scene (traffic models for driving simulators, fluid-mechanic models for the simulation of deformable objects and surfaces, pendulum models for the simulation of cranes and sloshing, etc). Since these models generate a high computational workload, their computation is isolated in some nodes of the cluster. Depending on the nature of the simulated process (fast-paced, slow-paced, etc), these nodes send the output of the physical models to the rest of the nodes of the cluster either in each frame or upon request [28]. In both cases, the bandwidth required by the application (generated by the update of few attributes of the elements in the simulation) is minimal. The possibility of decoupling simulation and visualization allows running IV applications on graphics clusters where their nodes are connected remotely. In these cases, the middleware must not perform poorly with the increment of

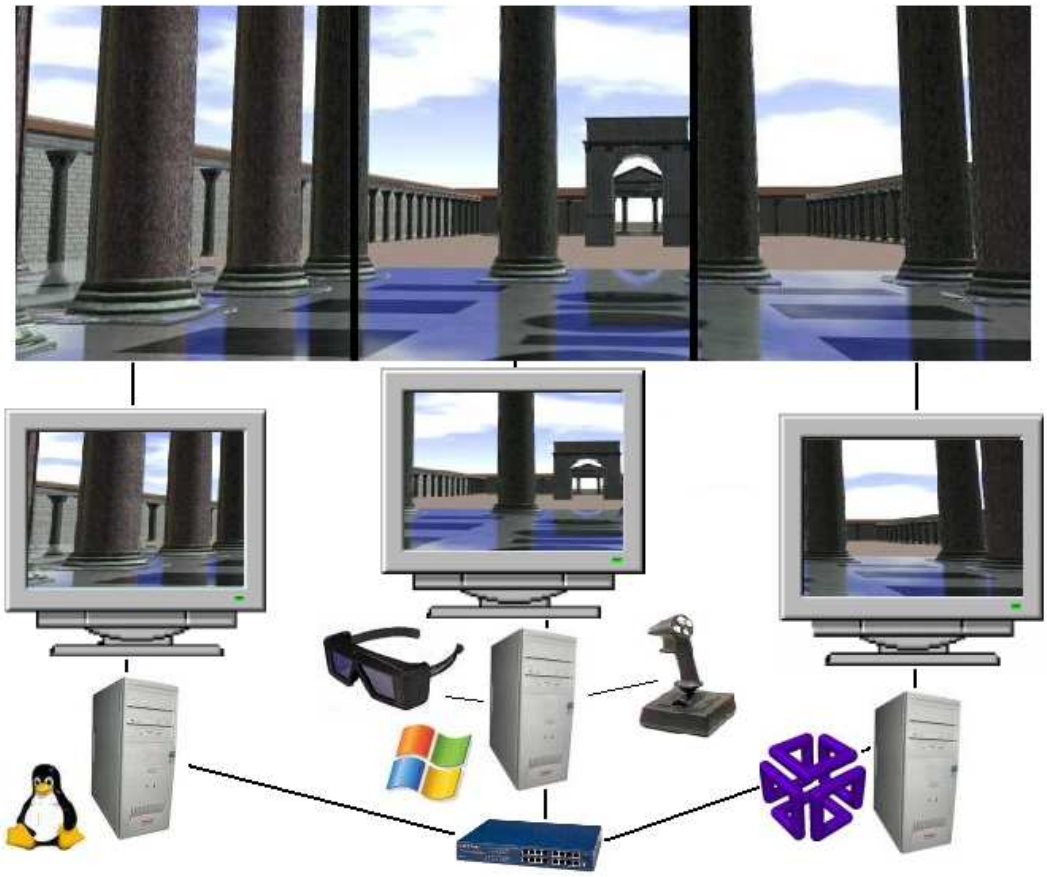

Fig. 1. Schematic diagram of an IV system based on an heterogeneous cluster

the communication network delay.

Several techniques based on cluster computing have been used to parallelize complex computations in high-performance computing (HPC) for many years [7,37]. Despite the fact that clusters offer an alternative to expensive supercomputers and can be used to drive multi-screen visualization systems, the existing parallelization techniques used for general purpose clusters (clustering techniques [23,32]) cannot be applied directly to graphics clusters. In this sense, graphics clusters add some constraints to virtual reality software. One of the most important constraints is the performance drop caused by the node synchronization mechanism. Since in IV systems the throughput is considered as the effective number of frames per second (denoted as FPS) drawn [37], the time spent for synchronization purposes decreases the system throughput as the number of nodes in the cluster grows. While these constraints are all solved by the hardware of shared memory computers, they must be solved by the software level in a graphics cluster [4]. This is the case of VR Juggler [3,5]. VR Juggler is a well-known open source suite of software tools that provides a multi-platform framework for the *development* and *execution* of VR applications [13,25,27]. Following other approaches [2,26,23,29,33,34,37], the kernel architecture of VR Juggler was modified in order to add clustering support [4]

to the features of the development environment.

Although some VR products based on cluster computing are currently available, to our knowledge no detailed proposal has yet been made about performance evaluation in immersive visualization systems. In this sense, existing contributions either lack of performance evaluation or show some results very far of the basic performance standards [37].

In this paper, we first describe the most important subsystem (for cluster computing) of VR Juggler's software architecture. This subsystem, called ClusterJuggler [4], enables the use of distributed and clustered computers for recreating immersive virtual environments. The main goals of ClusterJuggler are first, to adapt the cluster software to the particular hardware configuration of the virtual reality systems; second, to provide application portability and scalability from high-end shared memory systems to commodity clusters by hiding the clustering techniques from developers; and third, to allow users to customize their own software methods (based on cluster computing) being used to best meet their specific needs. Next, we present a global performance evaluation methodology for immersive visualization systems. Following basic and well-known performance standards [18,32], our methodology proposes the evaluation of system throughput with respect to both the network delay and the number of nodes in the cluster. This evaluation is performed by means of a reduced set of real IV applications that can be considered as graphics benchmarks in our methodology. The benchmarks are used in order to emulate the wide variety of levels of graphics and computational workloads generated by immersive visualization applications.

The rest of the paper is organized as follows: Section 2 describes the most important approaches to simulate multi-screen immersive visualization systems on a cluster of computers. Section 3 shows the modular architecture of ClusterJuggler (based on layers) and how can be extended the features of this platform oriented to support 3D real-time environments using a cluster of computers. Next, Section 4 presents the performance evaluation methodology designed for IV systems and Section 5 shows the performance results when this methodology is executed on VR Juggler. Finally, Section 6 presents some concluding remarks.

2 Motivation and Related Work

This section presents the current approaches used by cluster computing for supporting immersive visualization systems and how the performance evaluation has been conducted in order to evaluate them. Although several software libraries generate immersive environments by utilizing clusters of commodity

computers, it is possible to establish a classification of IV systems depending on how this issue is addressed: input data [2,16,34], remote shared memory [26], scene graph change lists [33,34], or graphics primitives [23,29,37].

In order for users to become fully immersed in a virtual environment, they must interact with it using one or more physical input devices such as a position trackers, 3D stereo glasses or gloves (see Figure 1). Since the objective is to provide the users with a sense of immersion, these devices obtain all the input needed to determine the changes in the state of the application. Solutions based on cluster computing use a software mechanism called *input data sharing* in order to start a distinct complete copy of the application on each node in the cluster. All input data are then synchronized across the cluster at the beginning of each frame loop. Thus, the application state remains consistent as long as it solely depends on input events. Despite the fact that input data sharing does not require any changes in the application relative to the memory management of computers based on shared memory architectures (since the application still has access to the same input data and rendering targets), mechanisms such as random number generation, consistent frame deltas, and start barriers are not addressed [23,32,5]. Examples of multi-screen immersive visualization systems based on cluster architectures are Net Juggler [2] and Syzygy [34]. While Net Juggler [2] uses message passing via the Message Passing Interface (MPI) [16] in its implementation, frameworks such as VR Juggler [5] and Syzygy [34] use the TCP/IP suite directly.

Remote shared memory approach offers another way to ensure that each node has an identical snapshot of that state for rendering each frame. Implementations of remote shared memory often require that the application programmer take special steps to use it. Special storage areas must be created, and in some cases, access to the shared memory must be controlled so that there may be multiple readers but only one writer. Different designs put more or less of the burden on the application programmer for understanding and managing these details. Implementations such as DIVERSE [26] are based on a shared memory architecture where a general inter-process communication programming tool guarantees identical copies among the nodes of the cluster.

Since most graphics applications are based on the manipulation of scene graphs [38], if one node keeps track of all changes made to the scene each frame, that node can send the changes to each of the other nodes to be reapplied to the local memory copy of the scene graph. Therefore, each node always has the information it needs to render an accurate version of the scene. This approach, called *scene graph change lists*, takes advantage of the fact that visual consistency and coherency is the critical aspect of all graphics clusters. OpenSG [33] and Syzygy [34] implement this method.

At the lowest level, all immersive applications generate a stream of graphics

commands that are delivered to the graphics hardware for rendering. This is accomplished by making calls to a low level graphics application programming interface (API) such as OpenGL. Software libraries such as Chromium [23] and DGL [29] are designed to intercept the *graphics primitives* for the rendering of each frame and to distribute them over a network in order to divide the rendering task among multiple nodes. This approach tends to require more bandwidth than any of the previously mentioned methods [37].

Despite all of these software libraries offering a wide range of solutions for developing immersive visualization applications, most of the contributions on this topic lack a rigorous study of the performance of the proposed approaches [18]. In some papers, authors just describe the architecture of the presented approach for IV systems based on cluster computing detailing a list of features of the presented product [26,29,33,34]. In these contributions any type of performance evaluation is neither considered nor presented. In other cases, authors show some graphical information about results of the execution of a VR application using the presented product [2,23]. In particular, the performance evaluation of Net Juggler is obtained using an interactive simulation (called NJFluid) based on a fluid flow model [2]. This simulation is an example of intensive pre-rendering computation and allows authors to show the performance drop of the IV system as the number of nodes in the cluster grows. A different type of immersive visualization application is shown in the performance evaluation of Chromium [23]. In this case, a volume rendering application uses 3D textures to store volumes and renders them with view-aligned slicing polygons, composited from back to front. Despite the fact that this framework for the development and execution of IV systems is tested using this highly graphics intensive application, the performance evaluation of Chromium shown in [23] is reduced to some execution snapshots of this testbed. Finally, a survey of performance evaluation results obtained from some software platforms for interactive cluster-based multi-screen rendering is presented in [37]. These performance evaluation results show the system throughput (in FPS) when a set of simple and well-known 3D models (Skeleton, Stanford Bunny and Dragon) are rendered on a two-by-two tiled display wall using a stand-alone computer and a graphics cluster composed of four nodes. Although these experiments show some network requirements of basic IV applications (in terms of average number of kilobytes sent per frame), obtained results are not correlated with any parameter of the cluster system such us number of nodes or network latency.

3 A new Cluster-Based Architecture for VR Juggler

Since the techniques presented in Section 2 have their own unique benefits and drawbacks, we present a modular and extensible architecture, called *Cluster-*

Juggler, that combines the advantages of all of them. ClusterJuggler's design contributes several key features not found in other architectures based on cluster computing for supporting IV systems: a layered architecture, run-time reconfiguration, and an extensible, component-based system.

The architecture of ClusterJuggler separates the aspects related to cluster computing for virtual reality into several layers. Each layer builds on the functionality of those below to provide additional features. This modular design allows us to implement and test each layer independently, and changes made to one can happen transparently to the layers above. ClusterJuggler uses the same advanced configuration scheme as VR Juggler [3,5]. In this configuration scheme, information arrives in the form of configuration elements. Basically, these elements are XML files and they are the fundamental units of configuration in VR Juggler [20]. Handlers of configuration elements are registered with an entity known as the Configuration Manager, and newly received configuration elements are delivered to the appropriate handlers. New configuration elements may arrive at any time during the lifetime of an application, thus allowing run-time reconfiguration of the software. Since Cluster Juggler is based on the VR Juggler architecture, it takes advantage of this feature [5]. ClusterJuggler allows nodes, displays, and input devices to be added, removed, or reconfigured as needed at run time. We have followed the traditional component-based approach for developing this architecture [39]. The code that uses the components is then responsible for loading implementations at run time based on some specification. Each component, called a plug-in, is a standalone module loaded at run time based on the user-specified cluster configuration. The plug-ins extend the ClusterJuggler core with specialized functionality based on cluster computing. Users can choose any of the plug-ins needed for their applications and their specific cluster configuration.

3.1 A Software Architecture Based on Layers

As Figure 2-a and Figure 2-b show, the architecture of ClusterJuggler has been designed as a set of components that are arranged into layers. At the lowest level, the Cluster Network provides a messaging interface for communicating with the entire cluster. The Cluster Plug-ins are built on top of the Cluster Network and provide the application developer with a set of components to construct the best solution for their applications needs. The top layer is the Cluster Manager, which acts as an interface to ClusterJuggler. Higher level code utilizes the Cluster Manager to control ClusterJuggler.

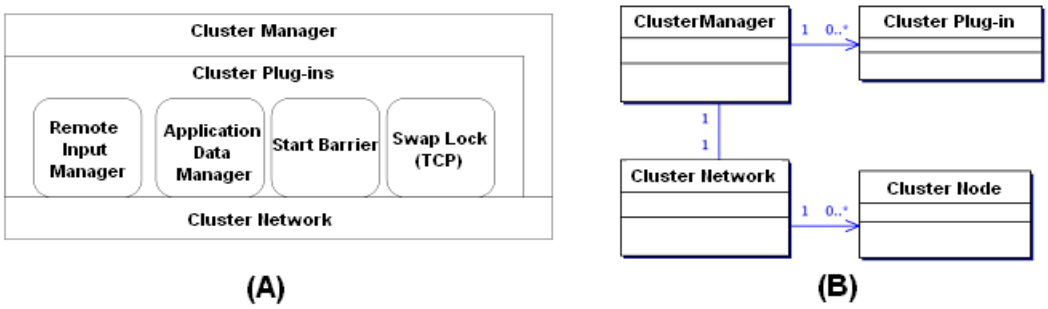

Fig. 2. The architecture of ClusterJuggler: (a) A layered view, b) a simplified UML class diagram

3.1.1 Cluster Manager Layer

This is the main layer in ClusterJuggler. This layer is responsible for handling the cluster configuration and for synchronizing the calls to each plug-in. Once all nodes of the cluster load the application code into memory, an entry point function is called to create an instance for each plug-in located in the Cluster Manager. At this time, each plug-in (selected by the user) becomes a mechanism which allows and defines the communication among the nodes of the cluster. In this sense, a cluster can incorporate a master/slave or a P2P network protocol depending upon the selected plug-ins. Since plug-ins can generate data inconsistency problems in the cluster, the Cluster Manager is responsible for making sure that all plug-ins have their run-time information dependencies satisfied during the simulation. In order to accommodate all possible needs, each plug-in has a well-defined interface and a contract that specifies the invocation timing. In order to guarantee a full compatibility between VR Juggler [5] and ClusterJuggler a micro-kernel architecture is adopted. In this case, at each step of the "kernel loop" the state of inputs and the graphics contexts are guaranteed by the Cluster Manager. The Cluster Manager can in turn invoke the methods of the plug-ins at well-defined times during the kernel loop.

3.1.2 Cluster Network Layer

This layer maintains an abstract representation of the system of interconnected nodes that comprise the cluster. This abstraction provides ClusterJuggler with a messaging interface for communicating with the entire cluster. Internally, it maintains a list of the nodes in the cluster along with the current network connections used to communicate with them.

3.1.3 Cluster Plug-ins

Cluster Plug-ins represent the point of extension for ClusterJuggler. This aspect of the design allows the addition of new plug-ins to address cluster-specific application issues not handled by the standard set of Cluster Plug-ins. By default ClusterJuggler incorporates the following plug-ins: the Remote Input Manager (*RIM*) plug-in, Application Data Manager (*ADM*), the Swap Lock (*SL*) plug-ins, and the Start Barrier (*SB*) plug-in.

RIM plug-in is responsible for distributing synchronized device data across the cluster. In order to ensure that all nodes in the cluster have a consistent snapshot of all input data, (regardless of the location of the physical hardware) RIM emulates each node of the cluster as a "device server". The device data is shared over the network using the platform-independent protocol provided by the Cluster Network layer. Users of the device server can take advantage of this idea and they can use input devices that might not otherwise be usable due to hardware or software limitations. This device location transparency allows not only the construction of a cluster from any combination of the platforms supported by VR Juggler [5], but also to balance the workload generated by a large number of VR devices by connecting them to separate computers.

ADM plug-in provides application developers with a method for sharing arbitrary application states across the cluster. This capability extends the fundamental input data sharing and demonstrates that the ClusterJuggler design allows multiple techniques based con cluster computing to be utilized in a single application. Sharing of application-specific data structures works by providing the application developers with a base class that they extend with their own type. The base class defines an interface for serializing and de-serializing the data structure. Application developers must implement this interface with serialization code that is specific to their data type. In order to ensure data consistency across the cluster, ADM not only maintains a different GUID (128-bit Globally Unique Identifiers) for each application specific data type [19], but it does not allow distinct nodes to have different copies of the same data. All the serialization function calls are performed in the same node of the cluster (which is configured by the user) called the "host node".

SL plug-in is used with the RIM Plug-in to ensure that all the applications running on the cluster nodes begin their execution on the same frame. SL plug-in creates a software barrier by sending signals between the cluster nodes. The plug-in uses a master/slave paradigm where each slave sends a signal to the master immediately before swapping the frame buffers. The master is identified through a configuration specific to the plug-in, and the remaining nodes are then identified as slaves. All the slaves then suspend their execution, waiting for the master to send a response signal. The master sends its response immediately before invoking the frame buffer swap operation. Upon receiv-

ing the response from the master, the slaves perform the frame buffer swap. Depending on how the interconnection nodes are configured, ClusterJuggler incorporates three different versions of SL plug-in: TCP swap lock, the parallel port swap lock, and the hybrid TCP/serial port swap lock [1].

Since each node in the cluster runs a distinct and complete copy of the VR Juggler application, ClusterJuggler needs a mechanism to guarantee that all nodes begin their execution on the same frame. This feature is provided by the *SB plug-in* using a master/slave paradigm similar to the SL plug-in. One node in the cluster is identified to be the master with a configuration specific to this plug-in type. The remaining nodes are therefore slaves. When each slave is ready to begin its frame loop, it sends a message to the master and waits for a response. When the master has received the messages from alle the slaves, it sends the responses to them. At that point, all nodes may begin their frame loop, thereby guaranteeing the goal of the SB plug-in.

4 Performance Evaluation of IV Systems

In computer graphics applications, the images are generated in real-time as they are being displayed to the user [9]. This is made possible due to the persistence of vision effect where images projected on the eye persist for about 40-50 ms. A sequence of still images depicting progressive stages of motion, when projected on the human eye at a rate closed to 30 frames per second (FPS), produces the perception of a continuous moving image. FPS is considered as the throughput in IV systems [37] and as this output value decreases below 30 units a smooth and steady 3D simulation becomes more jerky and jumpy. Basically, the generation of each new frame consists of two stages: a first stage where both the location of the geometries and the camera within the 3D virtual scene are updated, and a second stage where the rendered image is taken and placed into a display device [10]. While all the computation workload performed in the first stage is executed by the main processor in the system, most or all computations performed in the second stage are done by the graphics processor using its onboard memory. Since both stages can become a bottleneck depending on the requirements of the applications and, therefore, limit the performance, the throughput of a graphics application depends on the performance and capabilities of both the main and graphics processors. When bottlenecks arise the throughput decreases significatively below 30 FPS (even for an IV composed of a single node) and the system is said to be saturated.

Taking into account these considerations, an IV system based on cluster computing should be considered as an efficient tool when as the number of nodes increases the throughput must not go down under 30 FPS, even if the nodes

are connected remotely [22,28]. Additionally, if the IV system enters the saturation point, the performance drop in the system throughput should be as tiny as possible in order to restrain the jerky effects of obtained frames. Therefore, a performance evaluation should be conducted in order to ensure the efficiency of a given IV system. Unlike other types of systems based on clusters of interconnected computers [18], the evaluation methodology for immersive visualization systems is not standardized.

In this section, we propose a new performance evaluation methodology for immersive visualization systems. This methodology consists of investigating the performance drop in the system throughput by analyzing the effects of varying both network latency and the number of nodes in the cluster. Moreover, this methodology includes the use of a benchmark specification in order to implement VR testbeds ranging from highly graphics intensive to highly computational intensive applications.

4.1 A New Performance Evaluation Methodology for IV Systems

In order to show the performance of ClusterJuggler, we present a candidate for a standard performance evaluation methodology for immersive visualization systems. Without trying to establish a full performance evaluation model, and since to our knowledge no relevant proposal have been done in this topic, this methodology evaluates the throughput of IV systems as a particular type of distributed system. In this case, the distributed system is composed of different computers that can be operated simultaneously to generate a seamless 3D virtual scene.

The proposed evaluation methodology consists in measuring the obtained throughput by the IV systems (in terms of FPS) as both the number of nodes (maximum graphics resolution) and the network delay increase. Unlike other proposals [2,23,37], these correlation studies are obtained by means of a set of applications that try to cover the spectrum of graphics-intensive and computational-intensive workloads. The aim of this methodology consists in analyzing the behavior of a given IV system in function of the two workload parameters which limits the maximum throughput of the system. The advantage of this methodology is that it allows us to collect actual information about the performance of a given IV system in order to either compare the global performance of different IV systems, or select the most proper cluster computing environment (such as VRJuggler, Syzygy, Chromium, etc) to execute a given IV application on a system. Since this selection is based on a workload criterion, the methodology includes a benchmark specification which addresses the issue of evaluating the throughput in IV systems. The benchmark specification recommends to obtain the correlation studies by means of four different

Table 1
Description of the benchmark specification proposed as a part of the methodology

	Computational Workload	Graphics Workload
APP1	LOW	LOW
APP2	LOW	HIGH
APP3	HIGH	LOW
APP4	HIGH	HIGH

applications according to different levels of generated workload. This specification is composed of four types of applications (denoted from APP1 to APP4) that recommend to execute the different combination of graphics-intensive and computational-intensive workloads. The Table 1 shows the proposed relationship between the computational and graphics workload for the four type of applications included within the methodology.

Because of the heterogeneity involving the IV systems based on cluster computing (described in Section 1), the proposed methodology does not focus on a certain type of platform or architecture in order to be considered as a good candidate for a standard performance evaluation methodology for IV systems. Moreover, the included benchmark specification tries to be clear, self-contained, and as short as possible [35], avoiding any particular implementation or programming technique.

5 The Performance Evaluation in ClusterJuggler

In order to show the performance of ClusterJuggler and the flexibility of the proposed evaluation methodology in cluster systems, we present the evaluation results following two different approaches. Although both approaches follow the proposed evaluation methodology for IV systems, each of them focuses on different aspects related to the level of abstraction at which the performance evaluation for this type of system can be considered. In this sense, an *overall* approach shows how the performance drop appears (in terms of FPS) when some parameters in the cluster system are varied. On the other hand, and since this performance drop is caused by the synchronization mechanisms of cluster nodes, we have divided the kernel of ClusterJuggler attending to the type of performed operations. Once the code division has been accomplished, a *module* approach shows the way in which the synchronization time measured by the overall approach is consumed by the different kernel operations. It is

worth mention that while an overall approach can be applied to any type of IV system regardless of its software architecture, a module approach requires not only to monitor the graphics and network operations executed by the user application, but also the operations performed by the kernel of the IV system. Since not all the frameworks for the development and execution of IV systems are released as open source software (as VR Juggler is), if the access to the source code of their kernel software is not possible [23,26,29], then it is disabled to perform a performance evaluation following a module approach.

Since the goal of performance studies in this kind of distributed systems is to predict the effects that the cluster parameters have upon the rendering time of immersive visualization applications, we have selected a group of representative immersive visualization applications as the graphics benchmark. This set of application tries to cover the spectrum of graphics-intensive and computational-intensive workloads of IV applications and follows the recommendation of our benchmark specification (proposed as a part of our methodology) described in Table 1. Although the graphics benchmarks could be implemented from the scratch, most of the frameworks for the development and execution of IV systems include several demonstration application that could fit the proposed benchmark specification. Concretely, in the case of VR Juggler, we have chosen four applications based on the proposed criteria: "*cubes*", "*agua*", "*hindu*" and "*mpapp*". These actual applications follow the recommendations described in APP1, APP2, APP3 and APP4, respectively.

"*Cubes*" is an extremely simple application where 1000 cubes are drawn floating in the space. This simplicity allows the "*cubes*" application to generate low levels of workload in terms of both graphics and computation requirements. The next application, "*agua*", takes advantage of special hardware techniques, such as vertex shading [14], in order to recreate a real-time travel around a complete deep sea reef. Despite the fact that the computational workload generated by "*agua*" is very low, the huge use of the graphical card capabilities allows this application to be considered as highly graphics-intensive. The third application, "*hindu*", is a virtual walkthrough which allows users to explore the Radharaman Temple (Vrindavan, India). This application uses a set of animated virtual characters in order to perform a traditional religious ceremony inside of the temple. "*Hindu*" is not only graphically intensive as it contains large amounts of polygons and textures (for both temple and characters), but also it is computationally intensive due to the time it takes to generate the movements and shadows for all the characters in each frame of the simulation. Finally, on the opposite extreme of the application spectrum, "*mpapp*" performs a real-time simulation of a square piece of cloth which has been modelled as a simple mesh surface. Since this mesh is generated by means of a complex 3D polynomial equation, "*mpapp*" requires a minimal graphical workload but it is highly computational intensive. Figure 3 depicts different snapshots of the four proposed VR applications taken when they are executed in a stand-

alone configuration on VR Juggler. All the applications use OpenGL (with any type of graphics optimization or advanced tool to speed-up rendering) as an average programmer would use it.

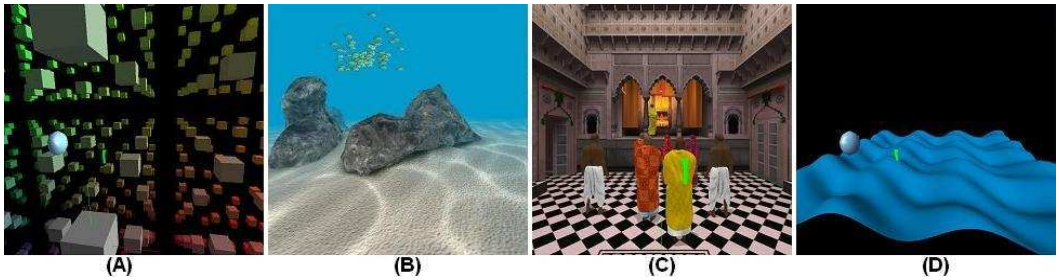

Fig. 3. Snapshots obtained from VR Juggler for: (a) Cubes (b) Agua (c) Hindu and d) Mpapp

Although an important issue when analyzing the performance of cluster systems is how network bandwith/latency and system throughput are related [18,31,32], this concept is not taken into account when virtual reality cluster systems are analyzed [26,29,33,34]. For this reason, we propose the performance evaluation of immersive visualization systems based on cluster architectures as a study of the relationship between the system throughput and both network latency and the number of nodes in the cluster.

5.1 Characterization Setup

VR Juggler includes a portable runtime library (called *VPR*) designed to provide a cross-platform, object-oriented abstraction layer to common operating system services [3,5]. Among other features, VPR includes the ability to acquire performance statistics in order to analyze the performance (in terms of execution time) of the different parts of virtual reality applications developed on VR Juggler. Basically, in a first step, this tool (called *VPR_PROFILE*) allows programmers to add some special tags in the source code. Once this monitorized code has been compiled and executed, VPR generates an output report containing the average time taken to execute the C++ functions (in milliseconds) where the tags were inserted.

Since VR Juggler is open source software, we have performed not only the decomposition and monitorization of the VR applications included in the proposed graphics benchmark, but also of the VR Juggler kernel. Following the presented performance evaluation methodology, this double decomposition and monitorization has been necessary to measure the impact of the variation of network latency and number of cluster nodes. The basic idea of this monitorization consists in measuring not only the system throughput, but also analyzing and isolating the delay generated by the synchronization mechanism located in the kernel of the IV systems.

Figure 4 shows an example of this decomposition for "*hindu*" application. A similar decomposition has been accomplished for the rest of the applications contained in the proposed graphics benchmark. This figure shows how the time that application needs to perform a frame cycle has been divided in two different steps. These steps have been implemented in two different functions called "*Draw()*" and "*App()*". From the user's point of view, Draw() function is used to do all the drawing of the 3D scene. In this case, the scene of this VR application is composed of a temple (function RenderTemple()), a set of characters (function RenderAvatars()) and their respective shadows (function RenderShadows()). In App() function the application obtains the position and status of VR devices such as position trackers, 3D stereo glasses or gloves. These data are necessary to update the current user's point of view, prevent collisions and animate the characters located in the scene (function PreFrame()). As it is described above, the data acquisition process is transparent to the application developer and is performed by ClusterJuggler in the kernel of VR Juggler (function KernelJuggler()). Moreover, this kernel subsystem should synchronize the Draw() process in the next frame for all cluster nodes. In order to detect how the variation of the parameters defined in the proposed evaluation methodology affects to the delay generated by KernelJuggler() for these type of tasks, the kernel of VR Juggler has been broken down into eleven monitorized subfunctions.

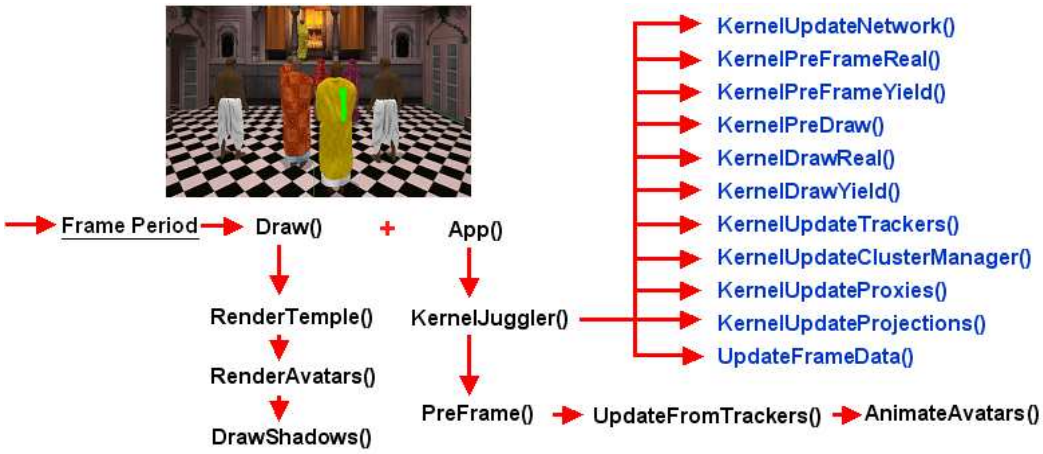

Fig. 4. Example of decomposition performed in "*hindu*" application

In order to analyze in detail the parameters included in the proposed evaluation methodology, we have taken advantage of a recent tool called *Netem* [21]. Netem (Network Emulator) is a multi-connection flow-level network emulator (derived from NIST-Net [8]) that can be used to emulate bandlimited links with fixed individual or total link capacity and/or transport latency. This general-purpose tool is based on a kernel module extension of Linux and has been used in order to study the effects of latency limitations on system performance and user interaction [30]. By operating at the IP level, Netem can emulate the critical end-to-end performance characteristics imposed by vari-

ous wide area network situations (e.g., congestion loss) or by various underlying subnetwork technologies (e.g., Ethernet, Fast Ethernet, cable modems). Basically, Netem allows each node of the cluster to ensure a non-uniform RTT (Round Trip Time) value for the transmitted TCP packets according to the specifications of the subnetwork technology. Since the correlation between RTT delay and the type of the physical network connection has been widely described in the literature of cluster computing [7,8], Netem becomes an excellent tool for emulating communication link delay in our test-environment hardware.

In all our experiments, the test-environment is composed by 8 Linux-PCs, each running RedHat 8.0 with a NVIDIA GeForce3 Ti200 (128 MBytes) graphics card, a 2GHz Intel Pentium IV Processor, 1 GByte of RAM, and 512 Kbytes of cache memory. The machines are connected to a Cisco Catalyst 3750 Gigabit Ethernet switch.

5.2 Performance Evaluation Results

This section presents the performance evaluation results of ClusterJuggler based on the proposed evaluation methodology. The goal of this performance evaluation is to identify and analyze the overhead introduced by cluster systems when virtual reality applications are executed in immersive visualization environments. This section does not discuss a detailed comparison of ClusterJuggler with other Virtual Reality software based on cluster computing for two reasons, mainly. Firstly, existing software packages differ in very dramatic ways. As Section 2 describes, immersive visualization clusters can synchronize on four different levels. The first level called "input data sharing" (at which VR Juggler operates) shares all input data among the nodes of the cluster and synchronizes them across the cluster at the beginning of each frame loop. At the time of writing, no other software packages is totally based on this approach (Syzygy [34] features a hybrid architecture based on "*input data sharing*" and "*scene graph change list*") to run immersive visualization applications on a cluster of commodity PCs. This total divergence of internal architectures makes impossible a comparison in depth of ClusterJuggler with other VR products based on cluster computing. Secondly, although [37] already seems to address this need, it compares many popular VR products based on cluster computing (including OpenSG [33], Syzergy, Chromium [23] and VR Juggler [3,5]) from the point of view of basic metrics for performance evaluation. This paper not only concludes that VR Juggler produces the fastest frame rates on all tests by keeping network traffic to a minimum, but also suggests that a performance evaluation comparing in detail several VR products from the application developer's point of view cannot be obtained as a whole.

Table 2
Performance drop for a C6 system in ClusterJuggler

	Stand-Alone	Cluster (C6)
Cubes	120.01 fps	67.12 fps
Aqua	76.83 fps	43.41 fps
Hindu	19.79 fps	18.69 fps
Mpapp	24.51 fps	20.73 fps

Table 2 shows this common performance drop (in terms of system throughput or FPS) when the same set of graphics benchmarks described in section 5.2 is executed on both a single computer and a cluster system by using ClusterJuggler. In this case, the cluster system has a classical configuration (called C6) composed of six nodes. This cluster configuration is usually associated with the most representative immersive visualization system, called *CAVE*, where users are closed into a six projection walls [3,20,23,24]. The results shown in this table indicate a significant difference, in terms of system throughput, when the same immersive visualization application is executed in different configurations.

Unlike in other approaches, the performance decrease shown in Table 2 depends on the type of application characteristics [2]. Therefore, the next sections show the variation of system throughput when each application is considered.

5.2.1 An Overall Performance Evaluation

The goal of our overall evaluation is to determine the influence of the most common parameters for the design of IV system on the global performance of ClusterJuggler. Since the system throughput in immersive systems is considered as the number of frames per second processed by the visualization system [37], we have measured this parameter following our performance evaluation methodology.

First of all, we have evaluated how the system performance varies for different cluster configurations. The results of these variations are shown in Figure 5. The Y-axis of this figure shows FPS values for the simulations performed with each system configuration. Each point in this plot represents the average value of the FPS obtained in each node after 25 executions of the same graphics benchmark. The standard deviation for any of the points shown in the plot was not higher than 4 FPS in any case. Figure 5 shows the values of FPS reached by ClusterJuggler depending on the number of nodes in the cluster.

This figure shows on the X-axis the number of nodes ranging from a C1 to a C8 configuration. While C1 runs the applications on a classical VR Juggler configuration composed of a single stand-alone node [5], ClusterJuggler executes the considered benchmark applications by means of eight synchronized computers in a C8 configuration. An example of C8 configurations are the curved-screen theaters composed of eight different projection walls that have become very popular in the real-time 3D simulators installed in many theme parks. Following the same parameters described in Section 1 for Figure 1, a C8 configuration achieves a 8192x768 seamless virtual monitor.

In this type of systems, the best performance (in terms of FPS) is achieved when the cluster is composed of a single node (called C1 configuration). In this case, the stand-alone computer does not have to spend time waiting for the synchronization with the rest of the cluster nodes (the performance drop is non-existent). Therefore, as more nodes are added to the cluster, the achieved resolution (graphics quality) of the immersive visualization environment is greater, but the overall FPS rate is lowered. This is the reason why the design of these systems is used to be considered as a threshold between throughput and graphics quality (in terms of achieved final resolution). Figure 5 shows that, for all the considered benchmark applications, FPS is almost linearly reduced as more nodes are added to the cluster. Moreover, the slope of the plot decreases with the workload generated by the application. In this sense, applications such as "*hindu*" or "*mpapp*" only have an average reduction of nine and seven, FPS, respectively when they are ported from a stand-alone system (C1) to a C8 configuration. The reason of this behavior is related to the linear network overhead added as new nodes are joined to the cluster system. This linear overhead is caused by the master/slave configuration of the SL plug-in described in Section 3.1.

Figure 6 shows the performance evaluation results obtained by ClusterJuggler when different values of RTT are considered in a C6 configuration. The first value on the X-axis (0.13 ms.) corresponds to the case in which no delay was added to the packets going out of the local Ethernet. This case shows the effective (and minimum) RTT value obtained in this configuration composed of six nodes and based on a Gigabit Ethernet backbone. In order to decrease the network throughput of the system, the rest of values located on X-axis correspond to the situations where Netem is used. Although the main goal of our study was to determine the performance of ClusterJuggler in LAN configurations, a wide range of delays is considered. This figure shows how FPS linearly decreases as communication link delay increases for all the considered benchmark applications. Unlike the above case, the FPS values tend to converge towards a similar threshold level when high latencies are emulated for all the benchmark applications. In this situations, ClusterJuggler spends most of rendering period waiting for the synchronization from both SL and SB plug-ins. In a situation that is closer to WAN environments, Figure 6 shows that

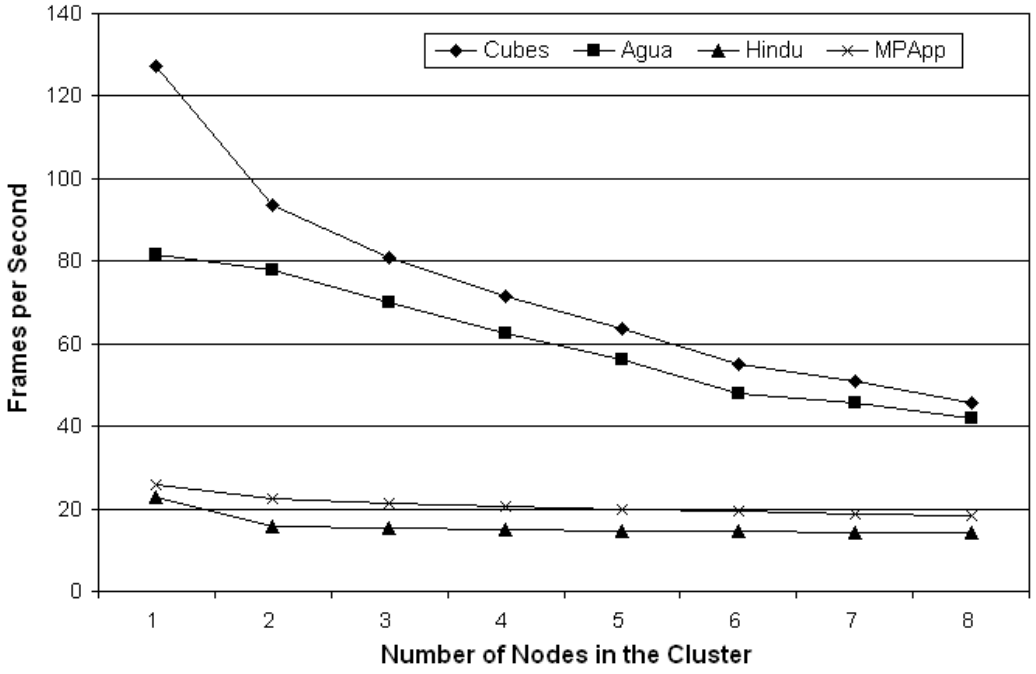

Fig. 5. Variation of system throughput with the number of cluster nodes

ClusterJuggler provides a very low reduction of performance levels, in terms of FPS, when VR Juggler immersive stand-alone applications are launched on commodity LAN clusters.

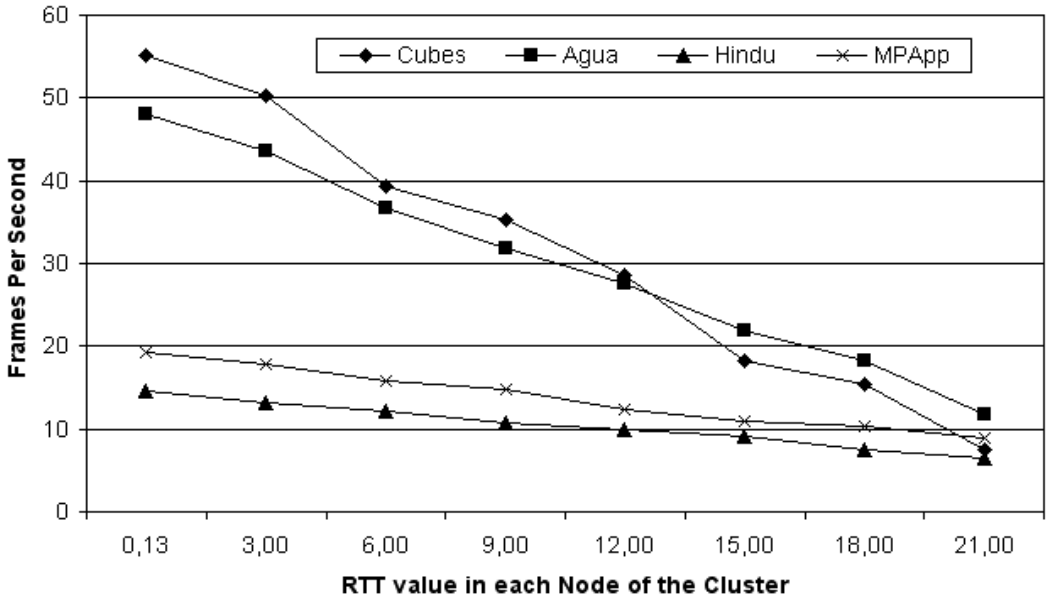

Fig. 6. Variation of system throughput with the network latency

Since values of RTT in these systems are not higher than a couple of milliseconds [7,11] and taking into account the definition of efficient IV system introduced in Section 4, these results show that ClusterJuggler can be considered as an efficient tool to simulate multi-screen immersive visualization

systems on a cluster of commodity computers.

5.2.2 A Module Performance Evaluation

Although obtained results in the overall approach show that ClusterJuggler is able to support cluster computing, this evaluation has been performed from the application developer's point of view. Because of this reason, we propose an additional performance evaluation following a modular approach. The goal of this new evaluation is not to show the characterization of performance drop in ClusterJuggler, but to analyze the behavior of the mechanisms inserted into the VR Juggler kernel by this software architecture based on cluster computing (RIM, ADM, SL, etc).

Following the kernel decomposition shown in Figure 4, we have monitorized the subfunctions in which the new kernel of VR Juggler has been divided. This monitorization was performed using VPR. As a result of this monitorization we have observed that only three of the eleven subfunctions of VR Juggler kernel (detailed in Figure 4) were responsible of the different obtained results when several configuration of IV systems on VR Juggler were examined. These subfunctions corresponds to KernelUpdateNetwork(), KernelUpdateProxies() and KernelPreDraw(). The remaining eight subfunctions of the VR Juggler kernel exhibited constant or very few variable execution times when the same graphics benchmark where executed on different system configurations. These configurations were based on the proposed evaluation methodology.

The code included in subfunction KernelUpdateNetwork() or *KUN* implements the features provided by ADM plug-in in ClusterJuggler. When KUN is executed, kernel access to the network in order to perform the serialization and de-serialization of data structures requested in this frame. Figures 7 and 8 show the variation of the executing time measured in this function as the values of the parameters presented in the proposed evaluation methodology (number of nodes and network delay) increase. The Y-axis of both figures shows the average value of the execution time of KUN function obtained after the same graphics benchmark drew 25.000 frames. Figure 7 shows the different values of execution times when different numbers of nodes were considered. It shows that for all the considered graphics benchmarks the time spent by ClusterJuggler in data serialization process increases linearly with the number of cluster nodes. The same behavior is shown for KUN function in Figure 8 when different levels of network delay have been recreated using Netem. In both figures, the slope of the execution time plot not only keeps more or less constant, but also is the same for both highly graphics intensive and highly computational intensive applications.

The operations associated with the RIM plug-in have been encapsulated in the

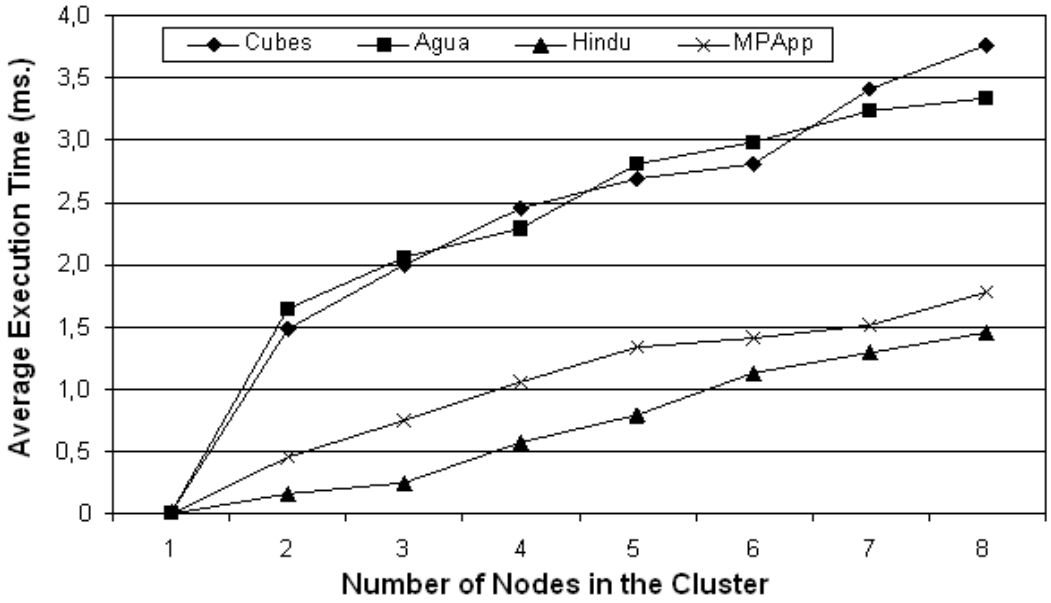

Fig. 7. Variation of average execution time in *KernelUpdateNetwork()* operation with the number of cluster nodes

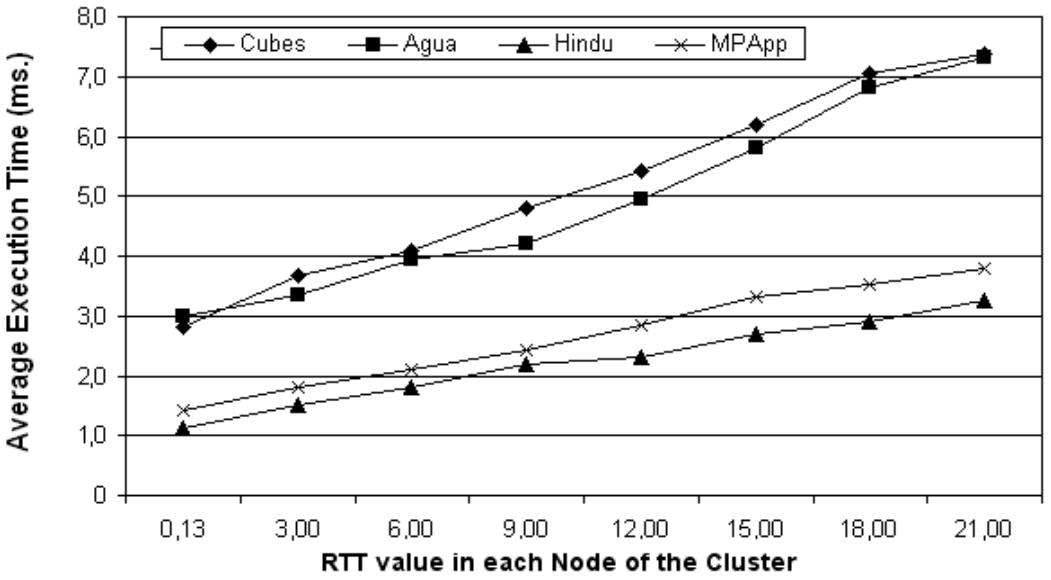

Fig. 8. Variation of average execution time in *KernelUpdateNetwork()* operation with the network latency

function denoted as KernelUpdateProxies() or *KUP*. In this function, nodes access to the rest of device servers in order to obtain data about the status of VR devices connected to the cluster system. Figures 9 and 10 show the average execution time reached by the KUP function when varying respectively the number of nodes in the cluster and the levels of network latency. Figure 9 shows the average time spent by ClusterJuggler kernel in KUP function as the number of nodes increases. Despite the fact that this figure shows a low impact in the values of execution time caused by the access of ClusterJuggler to

remote VR devices, this time increases significantly when the number of nodes is higher than six (C6). The reason for this behavior is due to the master/slave mechanism implemented in the RIM plug-in. When the number of nodes is high, the master node has to resolve an important number of requests from all slave nodes. Since the master node begins the software barrier process, slaves cannot render the current frame until all requests are processed and thereby the global system throughput is reduced. In the case of Figure 10, it shows a linear ratio between the relationship of the average time measured in KUP function and the network latency. Since Figures 9 and 10 show average values lower than those shown in the case of the KUN function (Figures 7 and 8) and taking into account the results described in Section 5.2.1, these results show that ClusterJuggler distributes VR devices among the nodes of the cluster with a low overhead in system performance.

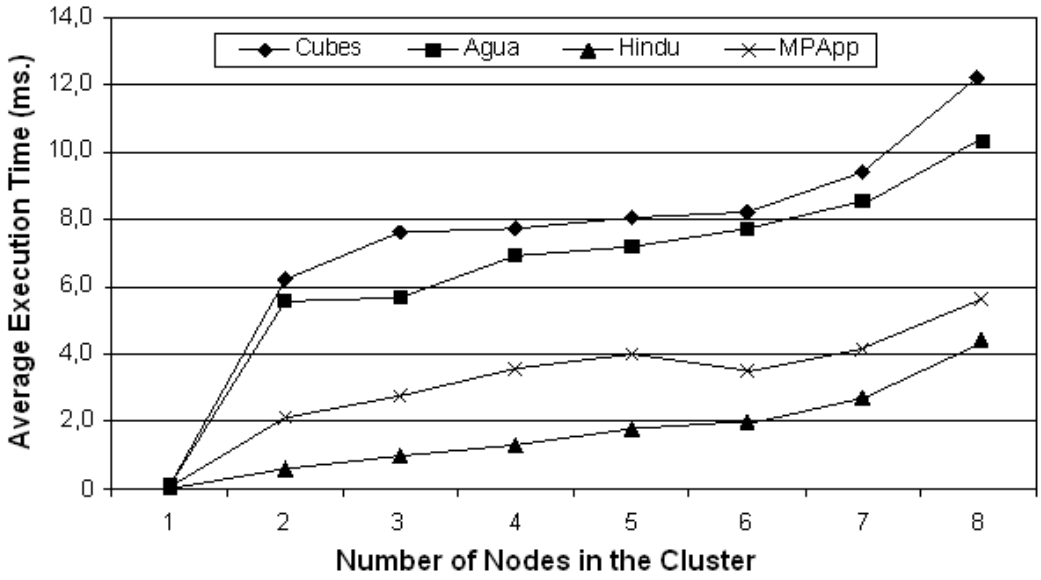

Fig. 9. Variation of average execution time in *KernelUpdateProxies()* operation with the number of cluster nodes

KernelPreDraw() or *KPD* implements the operations described in the SL plug-in in order to create a software barrier which ensures that all nodes draw simultaneously their segment of the 3D virtual scene in the next frame. In order to communicate the master node with the slave nodes, KPD follows a protocol based on a positive acknowledgment method. In this communication method, the master node sends a barrier message to the slaves which, in turn, reply it by means of an acknowledgment message. Figures 11 and 12 show the different values reached by the average execution time in the KPD code when the proposed graphics benchmarks have been executed on VR Juggler. Concretely, Figure 11 shows a non-linear dependence (approximately logarithmic) of the average execution time on the number of nodes in the cluster for all the considered benchmarks. This dependence is related to the way in which ClusterJuggler has implemented the protocol of message passing

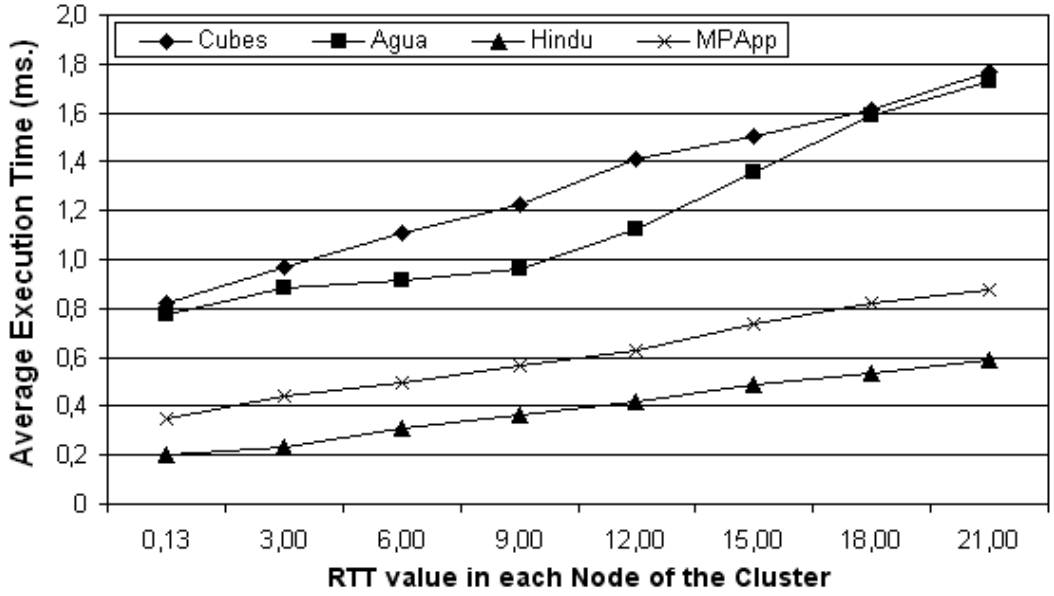

Fig. 10. Variation of average execution time in *KernelUpdateProxies()* operation with the network latency

that is used for starting and synchronizing the rendering of each new frame. Following the protocol described in Section 3.1, ClusterJuggler implements the response signals exchanged among the slave and the master nodes (barrier and acknowledgement) as simple messages of 1-byte length. In the case of the master node, the execution time taken by KPD corresponds from the instant the current frame is available within the frame-buffer memory of this computer (at the end of the function KernelPreFrameYield(), as it is described in Figure 4) to the instant when all the acknowledgement messages are received from the slave nodes. On the contrary, the execution time taken by KPD in a slave node is the time between the arrival of the barrier message (from the master node) and the transmission of the corresponding acknowledgement message. Due to the different volume of tasks performed by master and slaves nodes during the execution of this function, the execution time of the KPD function taken in the master node is significantly higher than the obtained values from the rest of nodes in the system. This is the reason why, unlike data serialization process implemented in the KUP function, the average execution time of the KPD function is not linearly related to the number of cluster nodes for the performed experiments. Figure 12 shows the values reached by the average execution time in the KPD when different values of network latency are generated. In this case, the execution time needed by ClusterJuggler for handling the software barrier depends linearly on the latency of the interconnection network.

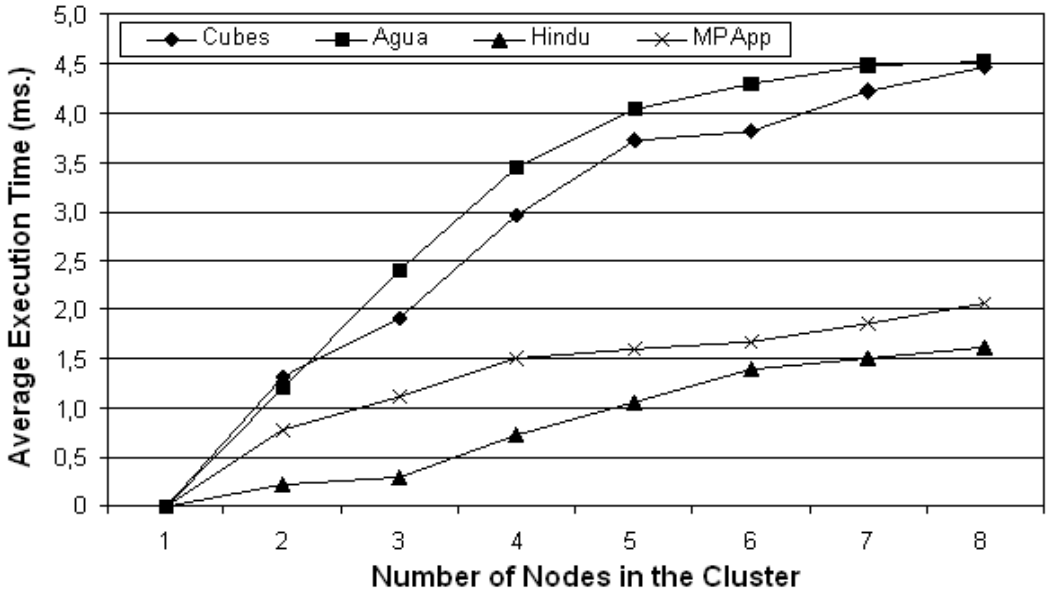

Fig. 11. Variation of average execution time in *KernelPreDraw()* operation with the number of cluster nodes

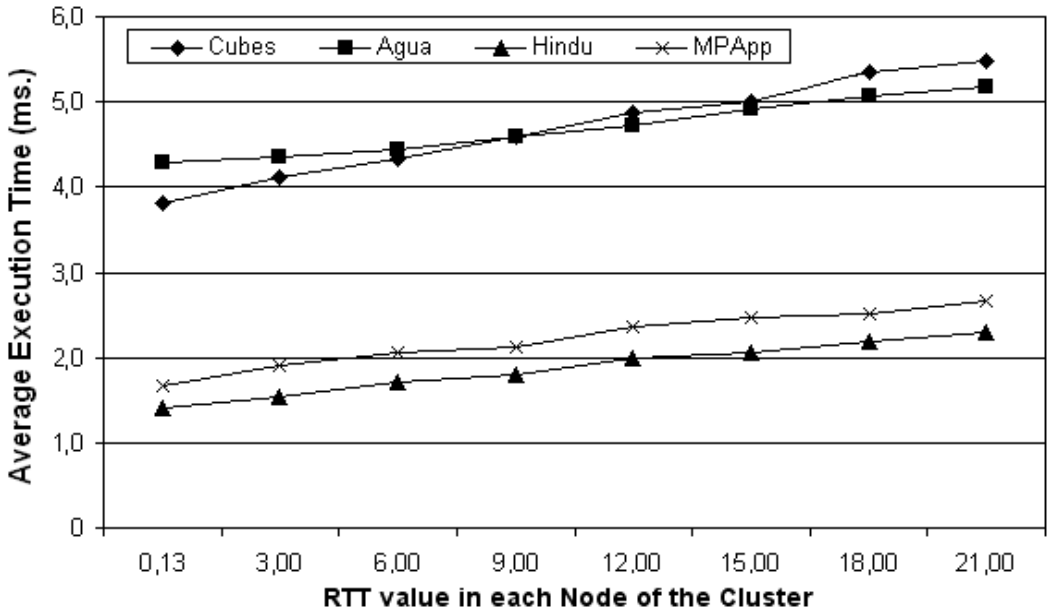

Fig. 12. Variation of average execution time in *KernelPreDraw()* operation with the network latency

6 Conclusions

In this paper, we have described the architecture and the performance evaluation of ClusterJuggler, an evolution of VR Juggler that allows one to design immersive visualization systems on cluster systems composed of commodity computers.

The open architecture of ClusterJuggler has been specifically designed to combine various existing techniques based on cluster computing. These techniques, offered at the hardware level or at the software level, allow developers of immersive visualization applications to meet their own specific needs. Since to our knowledge there are no standardized protocols for the evaluation of immersive visualization systems, we have proposed a new performance evaluation methodology for this type of distributed systems. This methodology proposes to evaluate the system throughput of immersive visualization systems, based on clusters of computers, with respect to both the network delay and the number of nodes in the cluster. Moreover, it proposes a benchmark specification in order to model the wide variety of levels of graphics and computational workloads generated by the most common immersive visualization applications.

The proposed evaluation methodology has been used in the performance evaluation of ClusterJuggler. The results of this evaluation show ClusterJuggler keeps the system throughput above 30 FPS as more nodes are added to the system independently of the LAN features. Furthermore, if the system enters saturation and is unable to fulfill this critical frame rate, ClusterJuggler minimizes the performance drop when the number of nodes increases in order to avoid jerky and jumpy 3D simulations. Taking into account all these properties, VR Juggler can be considered as an efficient software suite for the development and execution of IV applications.

In our case, ClusterJuggler has allowed us to migrate existing applications (designed initially for high-end shared memory computers) to cluster-based configurations while keeping high levels of frame-rate and without changes required to the application code.

References

[1] J. Allard, V. Gouranton, G. Lamarque, E. Melin, and B. Raffin. Softgenlock: Active Stereo and GenLock for PC Cluster. In *Proceedings of the Joint IPT/EGVE'03 Workshop*, Zurich, Switzerland, May 2003.

[2] J. Allard, V. Gouranton, E. Melin, and B. Raffin. Parallelizing Pre-rendering Computations on a Net Juggler PC Cluster. In *IPT (Intl. Workshop on Immersive Projection) 2002 Proceedings*, Orlando, Florida, United States, March 2002.

[3] A. Bierbaum and C. Cruz-Neira. Run-Time Reconfiguration of VR Juggler. In *Proceedings of IPT (Intl. Workshop on Immersive Projection) 2000*, Ames, Iowa, United States, June 2000.

[4] A. Bierbaum, P. Hartling, P. Morillo, and C. Cruz-Neira. Immersive Clustering with VR Juggler. In *International Conference in Computational Science*

and Its Applications (ICCSA-2005), number 3482 in LNCS, pages 1109–1118, Singapore, May 2005. Springer.

[5] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *IEEE Virtual Reality*, pages 89–96, Yokohama, Japan, March 2001.

[6] B. Bode, J.J. Hill, and T.R. Bejergerdes. Cluster Interconnect Overview. In *proceedings of USENIX 2004*, pages 217–223, Boston, USA, June 2004.

[7] R. C. Booth. A System Area Network Characterization In A Commercial Cluster. Master's thesis, Deptartment of Electrical and Computer Engineering, University of Minnesota, 1998.

[8] M. Carson and D. Santay. Nist Net: a Linux-based Network Emulation Tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, 2003.

[9] N. Chapman and J. Chapman. *Digital Multimedia, Edition 2nd.* Wiley, April 2004.

[10] K. Cok and T. True. Developing Efficient Graphics Software: The Yin and Yang of Graphics. In *Proceedings of ACM SIGGRAPH*, Los Angeles, USA, August 2001. ACM Press.

[11] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems, concepts and design, Edition 4th.* Addison-Wesley, Pearson Education, June 2005.

[12] C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In *Proceedings of 20th ACM SIGGRAPH*, pages 135–142, California, USA, August 1993. ACM Press.

[13] J.A Dickerson, Y. Yang, K. Blom, A. Reinot, J.Lie, C. Cruz-Neira, and E.S. Wurtele. Using Virtual Reality to Understand Complex Metabolic Networks. In *Proceedings of the Atlantic Symposium on Computational Biology and Genomic Information Systems and Technology*, pages 950–953, September 2003.

[14] W. Engel. *Programming Vertex and Pixel Shaders.* Programming Series. Charles River Media, 2004.

[15] Z. Fan, M.M. Oliveira, C. Ma, and A. Kaufman. A Sketch-Based Interface for Collaborative Design Sketch-Based Interfaces and Modeling. In *Proceedings Eurographics Symposium 2004*, volume VI, pages 1–5, Grenoble, France, August 30-31 2004.

[16] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. *International Journal of Supercomputer Applications and High Performance Computing*, 8(3/4):165–414, Fall/Winter 1994.

[17] W.A. Gallus, C. Cervato, C. Cruz-Neira, G. Faidley, and R. Heer. A Virtual Tornadic Thunderstorm Enabling Students to Construct Knowledge about Storm Dynamics Through Data Collection and Analysis. In *13th Symposium on Education*, Seattle, USA, January 11-15 2004.

[18] E. Gelenbe. *System Performance Evaluation: Methodologies and Applications*. CRC Press, 2000.

[19] The Open Group. *DCE 1.1: Remote Procedure Call*. The Open Group, August 1997.

[20] P. Hartling, A. Bierbaum, and C. Cruz-Neira. Tweek: Merging 2D and 3D Interaction in Immersive Environments. In Nagib Callaos, Alexander Pisarchik, and Mitsuyoshi Ueda, editors, *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics*, volume VI, pages 1–5, Orlando, Florida, United States, July 2002.

[21] S. Hemminger. Network Emulation with NetEm. In *Proceedings of Australia's National Linux Conference (LCA)*, Canberra, Australia, April 2005.

[22] M. Houston. Designing Graphics clusters. In *Proceedings of IEEE Visualization Conference - Parallel Rendering Workshop*, Austin, Texas, United States, October 2004.

[23] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: A Stream Processing Framework for Interactive Graphics on Clusters. In *ACM SIGGRAPH 2002 Sketches and Applications*, Texas, United States, July 2002. ACM Press.

[24] J. Jacobson and M. Lewis. Game Engine Virtual Reality with CaveUT. *IEEE Computer*, 38(4):79–82, 2005.

[25] B. Karthikeyan, K. M. Bryden, and D. A. Ashlock. Visualizing Information Flow in Evolving Graph-Based Population. In *Proceedings of International Conference in Smart Engineering Design (ANNIE-2003)*, St. Louis, United States, November 2003.

[26] J. Kelso, L. Arsenault, S. Satterfield, and R. Kriz. Diverse: A Framework for Building Extensible and Reconfigurable Device Independent Virtual Environments. In *Proceedings IEEE Virtual Reality 2002*, pages 183–190, Orlando, Florida, United States, March 2002.

[27] C. Kim and J.M. Vance. Collision detection and part interaction modeling to facilitate immersive virtual assembly methods. *ASME Journal of Computing and Information Sciences in Engineering*, 4(1):83–90, June 2004.

[28] G. Kim. *Designing Virtual Reality Systems: The Structured Approach*. Springer, August 2005.

[29] K. Li, H. Chen, Y. Chen, D. W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J. P. Singh, G. Tzanetakis, and J. Zheng. Early Experiences and Challenges in Building and Using a Scalable Display Wall System. *IEEE Computer Graphics and Applications*, 20(4):671–680, 2000.

[30] J. Marsh, M. Glencross, S. Pettifer, and R. Hubbold. A Network Architecture Supporting Consistent Rich Behavior in Collaborative Interactive Applications.

IEEE Transactions on Visualization and Computer Graphics, 12(3):405–416, May-June 2006.

[31] A. Plaat, H. Bal, and R. Hofman. Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects. In *Proceedings 5th IEEE HPCA99*, pages 244–253, Orlando, Florida, United States, January 1999.

[32] X. Qin and J.L. Baer. A performance evaluation of cluster architectures. In *Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 237–247, Seattle, USA, 1997. ACM Press.

[33] M. Roth, G. Voß, and D. Reiners. Multi-threading and Clustering for Scene Graph Systems. *Computers & Graphics*, 28(1):63–66, February 2004.

[34] B. Schaeffer and C. Goudeseune. Syzygy: Native PC Cluster VR. In *Proceedings of IEEE Virtual Reality 2003*, pages 15–22, Los Angeles, California, United States, March 2003.

[35] S. E. Sim, S. Easterbrook, and R. C. Hol. Using Benchmarking to Advance Research: A Challenge to Software Engineering. In *Proceedings of the 25th International Conference on Software Engineering*, Porland, USA, May 2003. IEEE/ACM Press.

[36] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999.

[37] O. G. Staadt, J. Walker, C. Nuber, and B. Hamann. A Survey and Performance Analysis of Software Platforms for Interactive Cluster-Based Multi-Screen Rendering. In *Proceedings of the Workshop on Virtual Environments 2003*, pages 261–270, Zurich, Switzerland, 2003. ACM Press.

[38] P. Strauss and R. Carey. An object-oriented 3d graphics toolkit. In *Proceedings of 19th ACM SIGGRAPH*, pages 341–349, Chicago, USA, August 1992. ACM Press.

[39] C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object Oriented Programming*. Component Software Series. Addison-Wesley Publishing Company, New York, NY, second edition, 2002.

[40] T.M. Wasfy and A.K. Noor. Visualization of CFD results in Immersive Virtual Environments. *Advances in Engineering Software*, 32:717–730, 2001.

[41] D. Wright. Survey of projection-based immersive displays. In *Proceedings of SPIE*, volume 3957, pages 482–492, California, USA, May 2000. Electrohome Editorial.