MuSA: a scalable multimedia server based on clusters of SMPs

Roberto Canonico, Giuseppe Capuozzo, Giulio Iannello and Giorgio Ventre
Dipartimento di Informatica e Sistemistica
Università di Napoli v. Claudio, 21 – 80125 Napoli
{canonico,capuozzo}@grid.unina.it,{iannello,giorgio}@unina.it

Abstract

Clustering is a new paradigm for high performance computing. So far cluster technology has been developed for parallel computing applications, but still many open issues exists in areas like parallel I/O, scheduling and QoS support. These issues are important for non-computing intensive applications like multimedia servers. In this paper we present an architecture for this kind of servers based on a cluster of commodity off-the-shelf SMPs. Relevant features of our design are high scalability and reconfigurability to deal with expandable and heterogeneous hardware setups, integration of the mass storage system and the network, low latency and responsiveness, mechanisms for QoS support. We also report some experimental results obtained by the implementation of the server on a Myrinet cluster of dual-Pentium machines.

1 Introduction

Clusters have been recently defined as "a type of parallel or distributed systems that consist of a collection of interconnected whole computers used as a single, unified computing resource", where *whole computer* is meant to indicate a normal, whole computer system that can be used on its own: processor(s), memory, I/O, OS, software subsystems, applications [7]. Clusters are an interesting alternative to massively parallel processors (MPPs) since they offer: scalability because additional processing power can be easily provided by adding computers to the cluster; availability by allowing servers to "back each other up" in the case of failure; manageability by providing a "single system image" to the user of the cluster.

Another interesting feature of clusters is that they are usually made up by commodity off-the-shelf (COTS) components, which make them very competitive with MPPs in terms of cost/performance ratio.

Achieving the single system image mentioned above, however, is a difficult task since common operating systems have not been designed to efficiently integrate the resources usually available in a cluster. So far, efforts in this direction mainly focused on making clusters an attractive alternative to MPPs for parallel computing applications. The main result of these efforts has been the development of low-level communication libraries capable of delivering to the applications the performance of modern Gigabit LANs [3, 4, 5].

Other issues, however, deserve attention to make cluster technology well suited to a wider range of applications. For instance, servers for multimedia applications, which greatly benefit from the mentioned features of cluster-based systems, would need further architectural support in areas different than high performance communications. A non exhaustive list of open issues in cluster computing which play an important role in multimedia servers includes: parallel I/O built atop standard file systems, interaction between high performance I/O devices of different nature (disks, network, multimedia peripherals), coordinated scheduling, memory hierarchy integration and caching, quality of service (QoS) support.

We have therefore started a project aiming to define, implement and evaluate a Multimedia Server Architecture (MuSA) which may take advantage of cluster technology. The server is mainly oriented to Video-on-Demand (VoD) services for their demanding requirements on the underlying system (e.g. high interactivity, dynamically changing workload). Main goals of our work are the definition and evaluation of mechanisms that help providing high scalability and reconfigurability to deal with expandable and heterogeneous hardware setups, integration of the mass storage system and the network, low latency and responsiveness for highly interactive applications, and mechanisms for providing QoS guarantees. These goals are pursued through the development of a working MuSA prototype and quantitative evaluation of architectural alternatives and trade-offs.

In this paper we present the design of MuSA and discuss its main properties with respect to scalability and reconfigurability. We then discuss our implementation strategy based on the High Performance Virtual Machine (HPVM) approach [3] that guarantees portability without compromising performance. We also report some preliminary experimental results obtained by running an implementation of MuSA on a Myrinet cluster of dual-Pentium. These results confirm the ability to scale up performances by properly configuring the MuSA system with respect to the available hardware resources. They also suggest that adaptive policies can be integrated into the architecture in order to provide graceful degradation of the QoS levels in presence of overloading.

The paper is organized as follows. In section 2 we describe MuSA and its main components. In section 3 we dis-

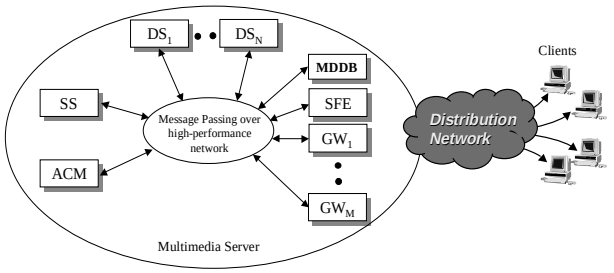


Figure 1: The Multimedia Server Architecture (MuSA).

cuss main implementation problems and give details on the hardware platform used for the experiments. In section 4 we present the experimental data and suggest areas for further research. In sections 5 and 6 we report about related work and conclude the paper.

2 The MuSA Architecture

The architectural design of MuSA is based on a functional decomposition of the server into well defined modules. Modules are then mapped on a properly configured cluster to provide scalable performance through task parallelism and module replication.

In designing a server architecture for VoD applications, three basic functionalities have to be provided: efficient data retrieval from the mass storage subsystem, isochronous data pumping to clients, and ability to handle the interaction with clients [8]. This motivated the organization of MuSA shown in fig. 1. The server is made up by a cluster interconnected by a message-passing high performance network (HPN). The server front end (modules SFE and GWs described below) is connected to the clients through a separate distribution network. MuSA consists of the following modules:

- The Meta-Data DataBase (MDDB), which provides clients with the full catalogue of available documents. Clients interact with MDDB when they start a session.
- Admission Control Module (ACM), which performs the admission control test and load balancing.
- The Server Front End (SFE), which interfaces clients and provides control functionalities during document playback. The SFE is assumed to provide these functionalities through standard protocols (e.g. RTSP).
- The Server Scheduler (SS), which maintains the status of the client sessions and periodically sends commands to the other modules to guarantee regular delivery of continuous data. The SS module also may perform adaptive control to provide graceful performance degradation when for any reason the server is overloaded.
- The Disk Server (DS), which performs physical access to its local storage subsystem and transfers data to the Gateway module (see below). This component essentially provides the coupling between disks and network. It may use several optimizations to improve I/O bandwidth including caching and concurrent access to multiple local disks. The DS module does not provide any timing control, but it relies on a SS module for isochronous data pumping.
- The Gateway (GW), which receives data from DS modules and transmits them to clients over the distribution network. This module essentially provides the coupling between networks of different nature. The GW is assumed to pack data according to standard protocols (e.g. RTP).

As in other video server prototypes [8, 2], in MuSA there is a clear separation between the Database server and the Video (continuous data) server, since these two subsystems have very different requirements. In a related project we are working on algorithms to index audio and video information so that sophisticated facilities for information retrieval can be added to the MDDB module [1]. Here, however, we are focused on the delivery of continuous data and we will not make further comments on this component.

The architectural design is based on decoupling the modules that deal with the three basic functinalities mentioned above. The main modules of the Video server are therefore the Server Scheduler, the Disk Server and the Gateway. The Server Front End and the Admission Control modules are auxiliary modules that play a role only in dealing with particular external events and do not affect directly the delivery of isochronous data.

All the main modules of MuSA can be replicated for performance and reliability reasons, leading to a scalable and highly available architecture. Parallelism can be exploited at many levels including task parallelism among different modules and data parallelism using module replication.

The hardware configuration which MuSA is assumed to be mapped on consists of a number of commodity workstations or PCs, interconnected by a Gigabit LAN. We assume also that at least part of the nodes are commodity SMP machines to deal with the performance requirements of DS and GW modules (see next section). The flexibility of this organization is twofold. On one hand the nodes of the cluster may be differently configured to meet the requirements of the modules composing a MuSA server. On the other hand, the replication of the modules can be tailored to the characteristics of the available hardware resources. This capability to support heterogeneity may be particularly useful to enable incremental upgrading, which is typical of cluster-based systems.

3 Implementation

We chose to implement the modules of MuSA as multithreaded processes interacting by message-passing. While multithreading is discussed in more detail below, the message-passing model was chosen for three main reasons: it is well suited to the streaming nature of the video server, it allows flexible mapping of modules to hardware resources, and high performance message-passing libraries are available for cluster environments.

Internally, MuSA modules have been designed in order to implement their functionalities in the most efficient way. For space reason we briefly discuss here only the core of the video server, namely the DS, SS and GW modules.

Two issues deserve particular attention: coordination between computing and multiple concurrent I/O activities in the DS and GW modules, and independent management of individual streams (allocation on disks, caching, timing, playback control) in the SS and GW modules.

As to the first issue, intra-module parallelism is achieved by discovering potential overlaps in concurrent activities and by mapping these activities onto different software threads.

In the DS module, data retrieval from disks and intra-cluster HPN accesses are performed by different threads, which may run in parallel on different CPUs. In our implementation, Disk Servers greatly benefit from the availability of more than one CPU per node, since the communication library we used (see below) tends to produce a CPU-bound workload. In the GW module, threads are used to perform in parallel the communications over the intra-cluster HPN and the external distribution network (fig. 1). The latter activity also includes protocol processing and other possible transformations on the outgoing video streams.

The problem of individual management of streams could be again solved by a generalized use of threads. This solution, however, has the drawback to potentially generate a huge number of concurrent activities that can degrade performance. Instead we chose a different approach based on the concept of *handler* which consists of an activity descriptor and a piece of code that operates on it. The SS and GW modules execute handlers associated to two kinds of events: deadlines (SS) and asynchronous message receptions (both SS and GW). Each handler performs actions regarding one stream and, before terminating, it associates a new handler to the next event concerning that stream. This event-driven solution takes advantage of the nature of stream management and greatly reduces the overhead with respect to a solution using one thread per stream and a general-purpose scheduler.

We consider portability a valuable characteristics for cluster-based systems. Hence the implementation of MuSA relies on a set of standard programming interfaces available on most cluster operating environments. All modules rely on three APIs providing support to message passing, access to the mass storage subsystem, and multithreading.

The Message Passing Interface (MPI) standardhas been adopted for intra-cluster communication. In particular, we used the VIRTUal System library (VIRTUS), a user level library implementing MPI we are developing in a related project. VIRTUS is a porting of the MPICH 1.1.2 implementation of MPI on top of the Fast Messages (FM) [3] messaging layer.

As for the disk I/O, the MuSA server does not need a special file-system, but uses standard I/O POSIX system calls, such as `fopen()`, `fread()` and `fseek()`, which access media documents as ordinary files.

Finally, as we have already mentioned, the MuSA modules are multithreaded processes. Unfortunately, multithreading is implemented in different ways in today's operating systems. To ease the porting effort, while deploying each model at its best, we have based our code on PTI, a Portable Threads Interface that we developed for threads creation, synchronization and management in different operating environments (Solaris threads, POSIX threads, Win32 threads). PTI is implemented as a set of macros which are translated, at compile time, in the corresponding OS-specific system calls.

The hardware platform we used for our experiments consists of a cluster of four SMPs, each of which is configured with two Intel Pentium-II 450 Mhz CPUs, PCI bus and 512 Mbytes of RAM. Each node is also equipped with one Ultra2Wide SCSI controller and three 9 Gbyte Seagate ST-39102LW disks. The intra-cluster high performance network is MyrinetTo transmit the multimedia streams to clients, the GW modules of the server use Fast-Ethernet network as an access network. All the nodes of the cluster are therefore equipped with two kinds of network interfaces: Intel EtherExpress Pro-100 Fast-Ethernet cards and Myrinet network adapters. The output bandwidth of GW modules can be increased either using multiple Fast Ethernet adapters or switching to Gigabit Ethernet technology.

Parameter	Value
VIRTUS/FM/Myrinet latency (μs)	$<$10
VIRTUS/FM/Myrinet bandwidth (MB/s)	100
Disk peak bandwidth per disk (MB/s)	17
Disk sustained bandwidth per disk (MB/s)	15

Table 1: Main parameters of the hardware software used.

In table 1 we report the main raw performance parameters characterizing our hardware/software platform. Disk performance has been measured transferring data in chunks of 1 MB. Disk peak bandwidth refers to consecutive accesses to a single very long file, while disk sustained bandwidth refers to interleaved accesses to many different files.

As an operating system, we use Microsoft Windows NT. This choice has been mainly motivated by the availability under Windows NT of a very high performance implementation of FM. However the use of standard interfaces makes our implementation easily portable. As a matter of fact, our implementation runs also on a Solaris-based cluster of Sun workstations we used for our first prototype. We are also working on a porting of the code to the Linux platform.

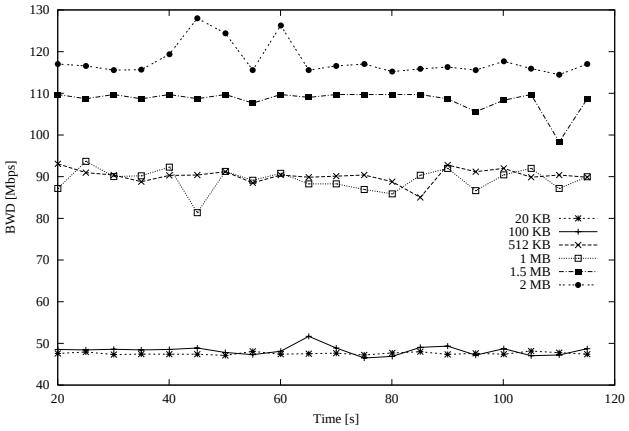
4 Preliminary experimental results

In this section, we report preliminary experimental results which establish the performance limits of MuSA on our platform in different configurations. The goal was to determine general guidelines to properly configure the server.

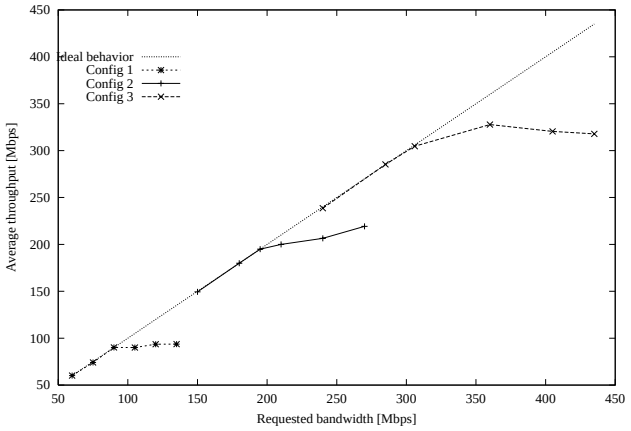
We focus on the server throughput, i.e. the maximum bandwidth (measured in Mbps) that the server can sustain with no service degradation. In all of the experiments that we report, we chose to handle with constant bit rate video streams. In particular, we used 1.5 Mbps streams, i.e. the typical bit rate of MPEG-1 encoded video streams. We assumed there is no service degradation if the average rate per stream is 1.5 Mbps and short-term peaks do not exceed $\pm$5% of the nominal value.

To concentrate on the effectiveness and scalability limitations of the cluster-based design, we decided to exclude the effect of data transmission over the distribution network. In practice, this means that the corresponding thread of the GW module just consumes locally the data streams. In all of the experiments that we ran, we verified that enough processing power was left available on the CPU on which this thread was allocated. The final release of MuSA will use such an extra processing power to perform the transmission of streams to clients using standard protocols (e.g. RTP, UDP, IP).

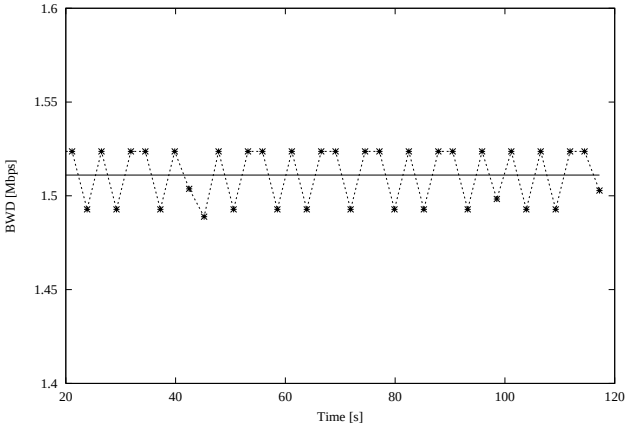
As we have mentioned in the previous section, MuSA consists of a number of software modules that can be mapped in different ways onto our SMP cluster. The impact of alternative mappings on performance was also object of our investigation. In particular, we report data concerning three different configurations, which use 1, 2, and 3 DS modules, respectively. Correspondingly, in the following we will refer to these configurations with numbers 1, 2, and 3. All cases use just one SS and one GW module. In all configurations but 3, there is just one module per node. In configuration 3, the SS and GW modules are mapped on the same dual-



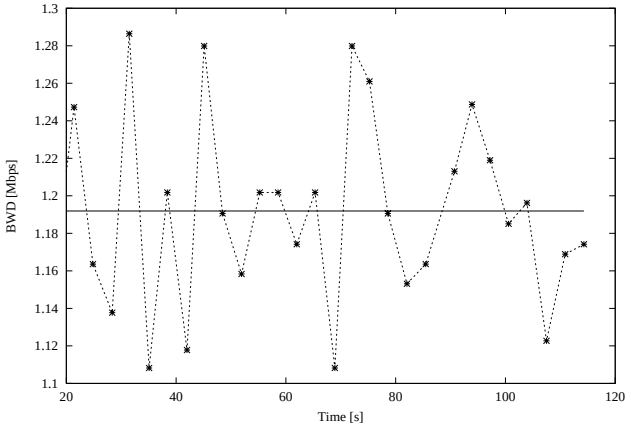
(a) Peak throughput vs. block size



(b) Average throughput vs. requested bandwidth



(c) Single stream bandwidth (no overloading)



(d) Single stream bandwidth (overloading)

Figure 2: Experimental results.

processor, while the DS modules are mapped onto distinct dual-processors.

The experiments consisted in generating a synthetic balanced load, by activating a number N of streams in the server and in comparing the requested bandwidth ($1.5\,N$ Mbps) with the average throughput delivered by the server.

As we expected, the server throughput depends on the retrieval *block size*, i.e. the amount of data that the DS module retrieves from the disk with a single read system call. Fig. 2-(a) illustrates the server throughput during a 100 s interval, for several block sizes. Of course, the larger the block size, the smaller the negative effect of seek and rotation overheads. However, the block size cannot be arbitrarily large because this tends to reduce the responsiveness of the system and increase the size of temporary buffers. From fig. 2-(a) it appears that a block size of 512 KB yields to significant better performance than smaller values, while for values larger than 512 KB the performance gain is not worth the latency negative effect. Hence, a 512 KB block size has been used in all of the following experiments.

Fig. 2-(b) reports the delivered throughput against the requested bandwidth, obtained with 1, 2 and 3 DS modules. The three curves are compared with the line representing the ideal behavior (delivered throughput equal to the requested bandwidth). It appears that in all of the three cases, the server throughput linearly increases up to a threshold value which coincides with the peak bandwidth of the I/O subsystem (about 90 Mbps per DS module). The threshold identifies the maximum number of streams that can be served at the 1.5 Mbps requested rate. Beyond the threshold value, the server throughput is limited by the DS performance. To confirm this interpretation we have instrumented the GW module to monitor the rate at which a randomly chosen stream is served. Fig. 2-(c) and 2-(d) report the actual bandwidth of a generic stream in a 100 s interval when the delivered throughput is equal to the requested bandwidth (no overloading) and when the delivered throughput is substantially less than the requested bandwidth (overloading), respectively. In the former case, the stream is served at a constant rate equal to the nominal bandwidth of 1.5 Mbps.

In the latter case, the rate at which the stream is served slightly varies in time and, as expected, it is in average less than the nominal bandwidth. However, even in this case the degradation is very smooth (20% loss in bandwidth, less than 10% variability). This suggests that the adaptive control mechanisms embedded in the SS module work properly. The introduction of different QoS classes could then be exploited to serve part of the streams at a guaranteed rate even under server overloading. We plan to explore this issue in future experiments.

Finally, from data concerning the CPUs utilization, we can draw the following conclusions. In the DS module, the thread accessing Myrinet through the FM library occupies as expected one CPU almost all the time, whereas the thread driving the disk leaves enough CPU time for at least another thread driving a second disk. This possibility will be explored soon in the attempt to remove the I/O bottleneck and better exploit all available resources. As to the SS and GW modules, their behavior shows that they are not bottlenecks even in configuration 3. This leaves room for integrating more sophisticated policies in the SS and the above mentioned protocol processing facilities in the GW.

5 Related work

Clustering, which is being used to develop high performance servers in application areas like data-mining, data-warehousing and terabyte storage, has also been proposed to support multimedia servers. However, in most cases clusters have been exploited almost like conventional distributed systems without special support to provide a single system image.

A project conducted at Syracuse University [8] explored a wide space of architectural alternatives for VoD systems, spanning from multiprocessors and massively parallel machines, to clusters of PCs. The MPP prototype was intended to support a single system image of the server through a shared disk architecture for the I/O subsystem. However, this attempt failed due to problems with the non-standard system software used. The final server architecture was a distributed architecture with multiple autonomous servers working together to assure scalability and load balancing.

The Tiger Fileserver proposed in [2] is a distributed, fault-tolerant real-time fileserver which can be used to transmit constant, guaranteed rate data streams. Tiger presents a controller node which is similar to our Server Scheduler. However there are several differences with respect to our proposal: QoS is achieved by strict resource reservation and there is no provision for adaptive mechanisms, VoD features are not fully implemented and, consequently, the front-end structure is much simpler than in MuSA.

Also the Elvira prototype [9] is a distributed video server, but its architecture does not include a synchronizing entity like our Server Scheduler. Rather, the equivalent of our DS modules work independently and their outputs are merged by an Integrator entity, which resembles our GW module. In any case, the elvira architecture is a network of essentially independent workstations, with no special support to clustering.

Finally, in [6] it is presented a software-based system for video effects processing. The authors investigate how spatial and temporal parallelism can be exploited to run such a compute-intensive application over a cluster of commodity computers. In a sense, this application covers complementary research areas with respect to our VoD prototype in that its workload requires a large amount of computation on a limited number of video streams.

6 Conclusions

In this paper we have presented MuSA, an architecture for multimedia servers based on a cluster of commodity off-the-shelf SMPs. We have reported experimental results which confirm the basic elements of our design and suggest interesting areas for further research. Besides adding more functionality to the prototype, issues that deserve further work in the short term include: increasing throughput of DS modules managing more disks per node; differentiating streams scheduling by means of priorities; using advanced communication services (e.g. RSVP) in the distribution network in order to extend the QoS guarantees up to the end user.

Acknowledgements

This work has been carried out partially under the financial support of MURST in the framework of the Project "Design Methodologies and Tools of High Performance Systems for Distributed Applications (MOSAICO)", and of the Consiglio Nazionale delle Ricerce (CNR) in the framework of the project ADESSO.

References

[1] G. Boccignone, M. De Santo, G. Percannella, "Joint Audio-Video Processing of MPEG Encoded Sequences", *Procs. of IEEE-ICMCS'99*, IEEE Press, Firenze, Italy, 1999.

[2] W.J. Bolosky, J.S. Barrera, R.P. Draves, R.P. Fitzgerald, G.A. Gibson, M.B. Jones, S.P. Levi, N.P. Myhrvold, and R.F. Rashid, "The Tiger Video Fileserver", *6th Int. NOSSDAV Workshop*, April 96.

[3] A.A. Chien, S. Pakin, M. Lauria, M. Buchanan, K. Hane, L. Giannini, and J. Prusakova, "High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance", *8th SIAM Conference on Parallel Processing for Scientific Computing (PP97)*, March, 1997.

[4] C. Dubnicki, L. Iftode, E. Felten, and K. Li, "Software support for virtual memory-mapped communication", *Procs. of the 1996 International Parallel Processing Symposium*, Aug. 1996.

[5] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: a user-level network interface for parallel and distributed computing", *Procs. of the 15th ACM Symposium on Operating System Principles*, Dec. 1995.

[6] K. Mayer-Patel, and L.A. Rowe, "Exploiting Temporal Parallelism For Software-only Video Effects Processing", *Procs. of the 6th ACM Multimedia Conference*, Sept. 1998.

[7] G. Pfister, *In Search of Clusters*, 2nd edition, Prentice Hall, 1997.

[8] M. Podgorny, G.C. Fox, "Video on Demand Technologies and Demonstrations", Tech. Rep., Northeast Parallel Architectures Center, Syracuse, March 1997.

[9] O. Sandsta, S. Langorgen, R. Midtstraum, "Video Server on an ATM Connected Cluster of Worksations", *SCCC'97*, Valparaiso, Chile, November 1997