Formal Verification and Analysis of Multimedia Systems

Sérgio Campos[1] Berthier Ribeiro-Neto[1] Autran Macedo[1,2] Luciano Bertini[1]

[1] Departamento C. Computação
Universidade Fed. Minas Gerais - BRASIL
{scampos, berthier, autran, bertini}@dcc.ufmg.br

[2] Departamento Informática
Universidade Fed. Uberlândia - BRASIL
autran@deinf.ufu.br

Abstract

Multimedia systems such as video-on-demand (VOD) servers are time critical systems. These systems have strict response times, which implies that a delayed response can have serious consequence. For instance, in the case of a VOD server, an immediate consequence of a delayed response time can be user dissatisfaction, what can ultimately lead to the end of a business based on this system. Therefore, analysis and verification of timing properties of multimedia systems is an important problem. To verify if time critical systems satisfy their time bounds, we discuss the use of formal methods tools, in the verification and analysis of multimedia systems. We have used Verus (a formal verification tool) to model and analyze the ALMADEM-VOD server, a component of a true video-on-demand system. The modeling of this server in Verus has provided great insight into its design and its dynamic behavior. Using the quantitative estimates provided by Verus, we have determined performance bounds to the server. These bounds have pointed out that the performance curve of the actual server was almost at the predicted upper bound (worst case) level. These curves have uncovered design inefficiencies. After optimizing the server, its performance has improved over 40%, showing how useful formal verification can be used successfully during the design of multimedia systems.

1 Introduction

The fast development of new technologies for high bandwidth networks, wireless communication, data compression, and high performance CPUs has made it technically possible to deploy sophisticated infrastructures, such as illustrated in Figure 1, for supporting a variety of multimedia applications [18, 29]. This type of infrastructure opens up opportunities for exploring multimedia applications such as quality audio and video on demand (from home), virtual reality environments, digital libraries, and cooperative design. The user has access to these applications through a laptop (connected, for instance, to a wireless link), a PC-based workstation, or a TV set connected to a set-top box. Due to the high interactivity of such applications, the success of the service is heavily dependent on the delays incurred in the high bandwidth network, on the delays incurred at the server and client machines, and on the performance at the multimedia server. To keep such delays within acceptable bounds, it is necessary to check the timing system properties.

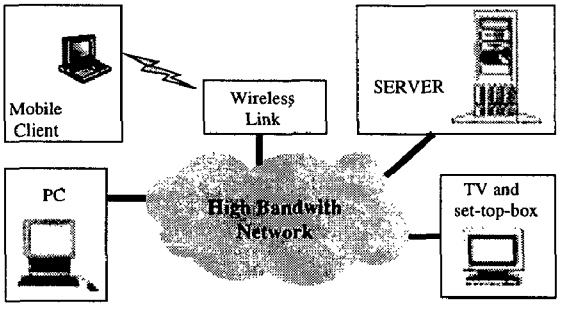

Figure 1: Overall architecture for a video service.

However, due to its size and complexity, checking if a multimedia system satisfies its timing specification

is an extremely difficult task. Traditional verification methods such as testing and simulation can not cover *all error possibilities*. An alternative approach to deal with this task is the utilization of formal techniques. Formal techniques can assure the correctness of a program with concern to certain desired properties. That is the approach we have used in this work.

The formal approach we adopt here is based on *symbolic model checking* (SMC) [4, 24], a formal method which has obtained significant success recently. This method allows the verification of timing properties of a system expressed as temporal logic formulas. Such formulas provide a foundation for the design and implementation of formal verification tools. One such tool is *Verus* [8, 5] — a software environment which can be used to verify time critical systems in general. In particular, in this work we use Verus to verify a specific component of the system — the multimedia server.

We focus our study on the verification of a low cost video on demand (VoD) server called ALMMADEM-VOD, developed within the ALMADEM[1] project. Such type of server is quite distinct from conventional servers, such as database and Web servers, which do not have to take into account strict time constraints. In a VoD server, failure to meet the time application constraints will certainly lead to user dissatisfaction and consequently to risks of commercial failure. Further, to be cost effective the VoD server must present good performance (which is usually measured as the number of users which can be served simultaneously).

We have modeled the ALMADEM-VOD server in Verus to verify its timing properties. We have also analyzed quantitative estimates provided by Verus for critical system parameters. These estimates have been compared with empirical results obtained from the server. This comparative analysis has led to a better understanding of the server operation and has revealed inefficiencies in the server. After inefficiency causes were eliminated, the system performance has improved over 40%. Formal verification has been very useful in detecting system inconsistencies and in enhancing our knowledge about this server.

This paper is organized as follows. In Section 2, we present related work. In Section 3, we discuss the Verus tool. In Section 4, we present the ALMADEM-VOD video server. In Section 5, we cover the modeling of this server in Verus. In Section 6, we present our analytical and experimental results. Our conclusions follow.

[1] ALMADEM (Analysis and Applications of Algorithms for High Performance Multimedia Networks) is a project financed by the Ministry of Science and Technology in Brazil, within the program PROTEM-III.

2 Related Work

Rate Monotonic Scheduling Theory (RMS) [23, 22] is a well known technique for real-time systems analysis. Given a set of processes defined by their execution time and frequency of execution, RMS is able to determine if the system is schedulable. However, RMS restricts the type of systems that can be verified, such as for example, distributed systems and systems with aperiodic processes (i.e., systems whose processes occur non deterministically).

Other works that also model a system as a state transition graph [17, 19], only determine if the properties are satisfied by the model, without providing any quantitative information. Other SMC-based techniques have been extended to deal with real-time systems [13, 30], but only determine if the properties are satisfied by the model, or provide limited quantitative information about the system [14].

Some verification tools adopt a continuous time approach [1, 21], contrary to Verus, which uses discrete time. Although continous time approach allows a more accurate measurement of time, it demands a larger space to represent the model. This demand makes modeling complex systems very difficult.

In formal tools based on Z [28], systems are specified in terms of rich mathematical structures like sets, relations, and functions. However, the proof of the specification generally is not totally automatic. In Verus the specification of a system can be totally verified in an automatic way.

Other approaches for verifying the correctness of temporal features of a multimedia application have been proposed in the literature. In [26], the temporal consistency of a hypermedia document is verified using an RT-LOTOS description, which is generated automatically from the document specification. A similar approach is adopted in [15] which proposes a synchronization model to verify the temporal consistency of a multimedia document. In [16], a suite of formal methods is used to verify the correctness of PREMO — a standard for the presentation of mul-

timedia objects. We observe, however, that such approaches are quite distinct from our work here which aims at verifying the dynamic temporal behavior and the performance of the multimedia system.

3 Verus

Verus is an efficient tool for performing the formal verification of multimedia systems and can exhaustively check the state space of systems with more than 10^{30} states, in a few minutes [7, 9]. Verus represents the system being verified as a *state-transition graph* and verifies properties about its behavior described in temporal logic. It also allows determining *quantitative information* about the system such as its reaction time to events and the number of times an event occurs in a given time interval. The information produced allows the user to check the temporal correctness of the model and also provides insight into the behavior of the system. Such type of insight can help the user identify inefficiencies and suggest optimizations to the design. Further, this analysis can be performed *before* the actual implementation, significantly reducing development costs.

Verus has already been applied to the verification of several large complex systems (which are in production or are components of current industrial products) such as an aircraft controller [9], a robotics controller [6], and a distributed heterogeneous real-time system [10]. In this work, we use Verus to verify the ALMADEM-VOD video server – a low cost VoD server implemented on a PC platform.

3.1 The Verus Language

The Verus specification language is presented in this section by a simple example. We discuss the modeling of a non deterministic event and an alarm (that rings when the event occurs). A Verus specification related to this scenario is presented in Figure 2. Statement 1 declares a boolean global variable. In Verus, the variable types are *boolean* and *integer*, and global variables are the means of communication among processes (each Verus function models an independent process). Statements 2 to 11 define the *alarm*, and statements 12 to 17 define the non deterministic *event*. The non determinism of the event is stated by the *select* statement (in line 14). It assigns randomly the value *false* or *true* to the variable *occurred*, each time it executes. The *wait* statements in the program serve two purposes: they determine the passage of time in the model, and they allow processes to observe the changes of value of all the variables of the model.

```
1 boolean occurred;

2  alarm () {
3    boolean ring;
4    while (true) {
5      if (occurred)
6        ring = true;
7      else
8        ring = false;
9      wait (1);
10   }
11 }

12 event () {
13   while (true) {
14     occurred =
           select{false, true};
15     wait (1);
16   }
17 }

18  spec
19  AG (occurred -> AF ring);
```

Figure 2: An example of a Verus program.

Properties of the system are expressed in *Computation Tree Logic* (CTL) [12]. A CTL formula is formed by atomic propositions, connectives ($\neg$ and $\wedge$), and/or temporal operators. These operators always come in pairs: a *path quantifier* followed by a *temporal quantifier*. A path quantifier states that a property should be true for all (**A**) paths from a given state, or for some (**E**) path from a given state. A temporal quantifier describes how the states should be ordered with respect to time, for a path specified by the path quantifier. Some examples of temporal quantifiers are: $\mathbf{F}\phi$, where ϕ holds sometime in the future; $\mathbf{G}\phi$, where ϕ holds globally on the path. Statement 19 is an example of a Verus property and informally means that "everywhere in the model, whenever the event indicated by variable *occurred* occurs (AG), the alarm will always ring sometime in the future (AF)".

3.2 The Model

The state transition graph generated for a program P is $G_P = \{S_P, I_P, T_P\}$, where S_P is the set of states, $I_P \subseteq S_P$ is the set of initial states, and T_P is the relation that defines the transition among states in S_P. Each state is labeled by the value of all variables of P; a state differs from others by the value of these variables.

The states of G_P can be represented by boolean formulas, and each formula represents the set of states in which the formula is true. For example,

the formula $(x \vee \neg y)$ represents all the states of G_P, in which x is true or y is false. T_P, the transition relation, can also be represented by boolean formulas. To do that we use two sets of distinct variables, S representing the current state and S' representing the next state. If the current state s is represented by the formula f_s over variables in S, and the next state s' is represented by the formula $f_{s'}$ over the variables in S', then the transition $T(s, s')$ can be represented by the formula $f_s \wedge f_{s'}$. As an example, the transition from the state (x, $\bar{y}$) to the state ($\bar{x}$, $\bar{y}$) is represented by the formula $(x \wedge \neg y \wedge \neg x' \wedge \neg y')$. T_P is the disjunction of all transitions in G_P.

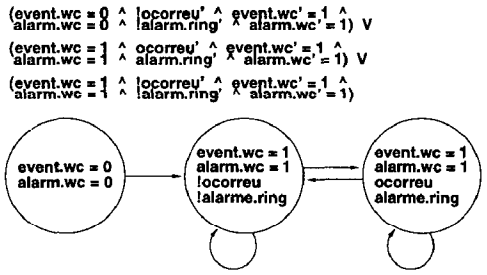

Figure 3: The transition relation and the related graph of the program event-alarm.

Figure 3 presents the transition relation and the related graph for the program presented in Figure 2. The variables *wc* are wait-counters that are used to *mark* the position of a wait statement in the program. The wait-counters determine the flow of the program execution, and an initial wait statement (wc = 0) is introduced by the compiler. An exclamation mark (!) before a variable indicates that this variable is false. A quote (') after a variable indicates the value of this variable in the next state. For example, the first transition says that in the current state the variables *event.wc* and *alarm.wc* are both equal 0 (zero); and in the next state the variables *occurred* and *alarm.ring* are both false. This transition is graphically represented by the edge from the state labeled by (event.wc = 0 alarm.wc = 0) to the state labeled by (event.wc = 1 alarm.wc = 1 !occurred !alarm.ring). The advantage of this representation is that it avoids the construction of the graph, representing only the conditions required by transitions, therefore minimizing the necessary space to represent the model.

Binary Decision Diagrams

As mentioned before, a Verus model is a state transition graph. However, the approach of modeling systems as a graph suffers from the state explosion problem. This problem is the exponential relation of the number of states in the model to the number of components of the system being modeled. Verus minimizes this problem because the model (graph) is represented by a Binary Decision Diagram (BDD)[3] that is a canonical representation for Boolean formulas . A BDD is obtained from a binary decision tree by merging identical subtrees and eliminating nodes with identical left and right siblings. The resulting structure is a directed acyclic graph rather than a tree. This allows nodes and substructures to be shared. The vertices of the graph are labeled with the variables of the Boolean formula, except for the two "leaves" which are labeled with 0 and 1. To ensure canonicity, a strict total order is placed on the variables as one traverses a path from the "root" to a "leaf." The edges are labeled with 0 or 1. For every truth assignment there is a corresponding path in the BDD such that at vertex x, the edge labeled 1 is taken if the assignment sets x to 1; otherwise, the edge labeled 0 is taken. If the path ends in the "leaf" labeled 0, then the assignment does not satisfy the formula, and conversely, if the "leaf" reached is labeled 1, then the formula is satisfied by the assignment. Figure 4 illustrates the BDD for the Boolean formula $(a \wedge b) \vee (c \wedge d)$. To each instantiation of the variables a, b, c, d which makes the formula true, there is a direct path from the node a to node 1.

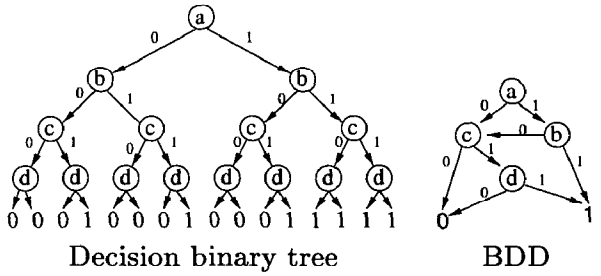

Figure 4: Decision binary tree and a correspondent BDD for the Boolean formula $(a \wedge b) \vee (c \wedge d)$.

3.3 Quantitative Algorithms

Most verification algorithms assume that timing constraints are given explicitly. Typically, the designer provides a constraint on the response time of some operation, and the verifier automatically determines if it is satisfied or not. Unfortunately, these techniques do not provide any information about how much a system deviates from its expected performance, although this information can be extremely useful in tuning the behavior of the system.

Verus implements algorithms that determine the minimum and maximum length of all paths leading from a set of starting states to a set of final states. It also has algorithms that calculate the minimum and the maximum number of times a specified condition can hold on a path from a set of starting states to a set of final states. Our algorithms provide insight into *how well* a system works, rather than just determining whether it works at all. They enable a designer to determine the timing characteristics of a complex system given the timing parameters of its components. This information is especially useful in the early phases of system design, when it can be used to establish how changes in a parameter affect the global system behavior.

Several types of information can be produced by this method. Response time to events is computed by making the set of starting states correspond to the event, and the set of final states correspond to the response. Schedulability analysis can be done by computing the response time of each process in the system, and comparing it to the process deadline. Performance can be determined in a similar way. In fact, the algorithms have been used to verify several real-time and non real-time systems [7, 9].

4 The ALMADEM-VOD Video Server

The fundamental premise in the development of the ALMADEM-VOD video server is that it should use only off-the-shelf low cost components, as also done in [18, 25, 29]. As a result, the server is implemented on a PC-based platform running the Linux operating system. To fullfill the real time requirements of the video application, the operating system is adapted in specific points such as the disk access and process scheduling routines.

The overall architecture of our video service is as illustrated in Figure 1. The user accesses a Web interface in a remote client machine (i.e., a TV with a set-top-box, a PC, or a laptop connected through a wireless link) to select a film (or object) of his preference. Once the film is selected, a requisition for a stream for that film is sent to the server (through a TCP connection). The server then runs an admission control routine which schedules the request if there are enough resources available (typically, the main bottleneck is disk bandwidth). Once the request is scheduled, blocks of data are sent periodically to the client in push mode (as UDP messages). Figure 5 illustrates the software organization of the ALMADEM-VOD video server.

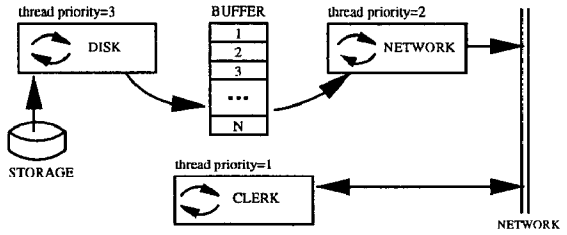

Figure 5: Software architecture of the video server.

Two data structures and three separate processes are distinguished. The data structures are called *storage* and *buffer*. The processes (implemented as POSIX *threads*) are called *disk*, *network*, and *clerk*. To ensure proper timing in the scheduling of these threads, we rely on one of the real time scheduling policies available with the Linux operating system. We use the SCHED_FIFO policy which implements a first-in-first-out scheduling scheme with static priorities. In this policy, the priority number of a thread is directly proportional to the system priority. Despite its simplicity, this scheme works quite well if the machine is dedicated to the video server task (i.e., we run the video server in run level 1).

The *storage* structure is composed of secondary or tertiary devices and is used to hold the collection of films available to the users. The current implementation of the ALMADEM-VOD server considers only secondary devices in the form of conventional SCSI-2 disks of 4G bytes each. The disks store the films encoded and compressed in MPEG-1 format. Each film is divided in *blocks* which are retrieved for delivery to the client machine. In its simplest implementa-

tion, which is adopted in this study, the ALMADEM-VOD server considers a *contiguous* layout of films on disk. In this layout, all blocks of a same film are stored contiguously on disk. More sophisticated layout schemes, involving striping techniques [2, 11], region-based allocation [20], and randomized placement [27], have been discussed extensively in the literature but are not the focus of this work.

The *buffer* structure is basically main memory space used to synchronize the *disk* and *network* threads. It is implemented as a circular buffer which is filled by the *disk* thread and emptied by the *network* thread.

The *network* thread is responsible for taking the blocks of film from the buffer and shipping them accross the network. It is scheduled whenever the *disk* thread is blocked at the disk driver waiting for a disk access to complete.

The thread named *clerk* listens at a TCP port for the requests from the client machines. Such requests might come from new clients or from a current client which requests, for instance, a *pause* in the exibition. Once it detects a client request, this thread passes the information to an admission control routine for proper scheduling. If the server is saturated (i.e., it is currently serving a maximum number of clients), a request for a new stream is not scheduled and a denial message is sent to the respective client.

The *disk* thread is responsible for reading the data from the secondary storage and storing them at the buffer area. To avoid delays introduced by the operating system (which we cannot control), disk accesses are performed through direct access functions which communicate directly with the SCSI controller device. For each active client, a separate block (which is composed of several MPEG frames) of (average) size B bytes is read, passed to the buffer area, and from there shipped (by the *network* thread) to the corresponding client machine. While that client consumes the frames in that block, other clients can be attended to. This *cyclic scheduling* process is repeated with a fixed time period equal to T, as illustrated in Figure 6. To implement this fixed time period, the *disk* thread monitors the Real Time Clock (RTC) device in the Linux kernel.

The time *period* T defines the *service cycle*. At each service cycle, all scheduled clients are served. To serve a client, the *server* incurs two fundamental delays: a seek time t_s and a transfer time t_r. The

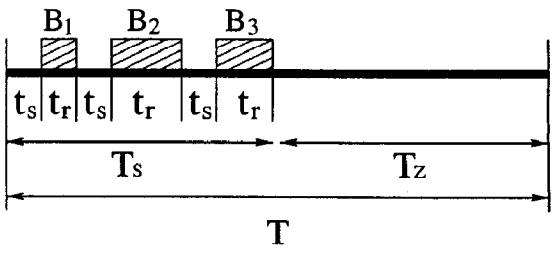

Figure 6: Service cycle with a duration of T seconds. t_s accounts both for seek and rotational delay times.

time t_s is the time to position the disk head at the proper cylinder, plus the time to position the disk head at the proper block of data in this cylinder (i.e., t_s includes rotational latency time). The time t_r is the time to read the block of data from disk. The total time T_s spent serving all clients in the system is called the *service time*. The *sleeping time* T_z is the portion of the service cycle in which no clients are served. Such sleeping time is necessary because, to avoid buffer overflow at the client machine, the server does not attend a same client twice in a service cycle. The ratio T_s/T defines the *occupation* of the service cycle. Larger the occupation of the service cycle, higher is the load in the system.

In the ALMADEM-VOD server (as seen in Figure 6), the transfer time can vary from one film to another because the block sizes, though constant for a same film, differ from one film to another. The reason is that the coding scheme might vary from one film to another (for instance, a film might be encoded for a smaller window size) and that the compression rate is not constant accross various films. The important detail is that, in the ALMADEM-VOD server, any block of any film is composed of roughly a same number of frames, which defines the duration of the service cycle. To exemplify, consider that each client consumes frames at the typical rate of 30 fps (frames per second). Then, if each block sent to a client includes 30 frames, the value of T is 1 second to avoid interruption in the continuous display of the film at the client machine. If each block includes 120 frames, then the value of T is 4 seconds.

To simplify the implementation, the ALMADEM-VOD server uses a Constant Data Length (CDL) block instead of a Constant Time Length (CTL) block [29]. The length of the blocks in which a given film is divided is determined by the maximum con-

sumption rate at the client. This ensures smooth display at the client machine. However, buffer overflow might occur at the client because the rate of arrival exceeds the average consumption rate. To avoid this problem, the client sends a *pause* message to the server whenever it detects that its buffer is filling up.

Let Rc_i be the rate (in bytes per second, or Bps) with which the *ith* client consumes a block of data. Further, let B_i be the size (in bytes) of the blocks of data for the *ith* client, as indicated in Figure 6. Then, the period T is given by

$$T = \frac{B_i}{Rc_i} \quad (1)$$

Additionally, let Rd be the transfer rate of the disks in our seconday storage and let N be the maximum number clients which can be served in a cycle. We can then write

$$\sum_{i=1}^{N} B_i = (T - N\ t_s)Rd \quad (2)$$

By substituting equation (1) into equation (2), we obtain

$$N = \frac{T}{t_s}\left(1 - \sum_{i=1}^{N} \frac{Rc_i}{Rd}\right) \quad (3)$$

Equation 3 shows that the maximum number of clients in the system is a direct function of the ratio $\sum_{i=1}^{N} Rc_i/Rd$ and thus, that the sum of all rates Rc_i must be smaller than the disk transfer rate R_d. Furthermore, the average block size (which determines the duration of the cycle service) must be large enough to provide an amortization of the time wasted with seek operations. In fact, a small average block size reduces the value of T making the fraction T/t_s smaller. This implies that the fraction of time available in a cycle for actually reading data from disk is smaller which leads to a reduction in the maximum number of clients which can be attended simultaneously.

Amortizing seek time (through an increase in the service cycle) is critically important because it improves server performance (in terms of the maximum number of clients which can be served). However, an excessive increase in the service cycle is counterproductive because it implies in an excessive *latency* — the time a new user waits to be served. This is because new users are only served in the cycle which initiates following their arrival. Additionally, large service cycles require larger memory buffers at the server and at the client machines.

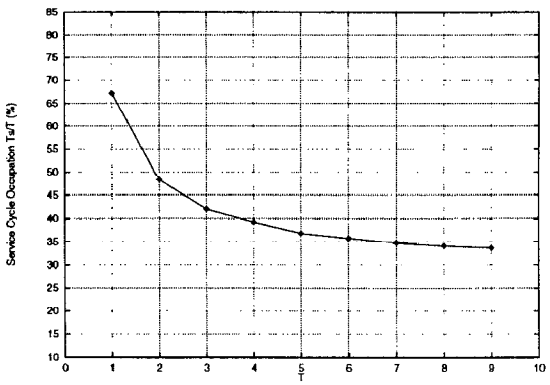

Figure 7: Service cycle occupation in the ALMADEM-VOD server for periods varying from 1s to 9s. The number of clients in the system is 20.

Figure 7 illustrates the occupation of the service cycle in the ALMADEM-VOD server for values of the period T varying from 1 second to 9 seconds. The number of clients in the system is 20. When $T = 1$ the occupation of the service cycle is high because of seek time (t_s). The system spends more time doing seek than reading block of films. When T is increased, seek time is amortized by the reading time. As shown, the amortization of the seek time is less significant after $T = 7s$ and does not improve much after $T = 4s$. For periods larger than 4 seconds, the server incurs higher latency without considerable additional gains in server performance. Thus, for the ALMADEM-VOD server, we adopt $T = 4s$ as a good compromise between client latency and server performance.

At each client machine, it is necessary to run the ALMADEM-VOD *client* module which is composed of three sub-modules: *network*, *buffer*, and *decoder*. The *network* component is implemented as a thread which receives blocks of film from the server and saves them in the *buffer* data structure. From the buffer, each block is passed to the *decoder* component through a Linux pipe. This architecture isolates the decoder (which is normally a commercial piece of software) from the client code (which we implemented) and provides for great flexibility. For instance, we were able to substitute the MPEG decoder for an audio player in a couple of hours.

5 Modeling the ALMADEM-VoD Server in Verus

Verus has a specification language that is similar to the programming language C. The Verus language provides special primitives that allow the user (i.e., designers and engineers) to model timing aspects of a system such as deadlines, priorities and time delays. Thus, modeling a multimedia system in Verus resembles writing a C program. A Verus specification is converted into a state transition graph (see Figure 3) and efficient search algorithm determines if the model satisfies the properties.

The modeling of the ALMADEM-VOD server has occurred while the designers of this system were developing it. The system parameters we have considered in the model were obtained from the version that was running at that time. The system parameters are: (1) a period of 4 seconds; (2) a consumption rate Rc_i at each client machine which varies from 1Mbps to 1.2Mbps (mega bits per second); (3) a film block size B which varies from 500K bytes to 600K bytes; (4) a contiguous layout of the blocks of each film in disk; (5) a disk transfer rate of 7.8MBps (mega bytes per second); (6) a disk seek time (including rotational delay time) which varies from 10ms to 20ms (milliseconds). The disk transfer rate were determined by reading disk blocks from disk. Although the nominal disk transfer rate of the disk is 10MBps, reading 500KB to 600KB disk blocks of films has led to disk transfer rate presented in item 5.

As mentioned before, performance in a VOD system is measured by the number of users it can serve simultaneously. The designers of the ALMADEM-VOD system had already measured the system performance, but they did not have any information about how close to optimum this performance was. Since the ALMADEM-VOD server is the bottleneck of the system, we have modeled it to determine the minimum and maximum number of users the server can support in a cycle of service (4 seconds). In this model (Figure 8), we have considered the seek and rotational delay times (t_s), and the transfer (t_r). Each *wait (1)* statement means 5ms. The time t_s varies from 10ms to 20ms (corresponding to 2 to 4 *wait's*) and is assigned non deterministically. The time t_r varies approximately from 60ms to 75ms (corresponding to 12 to 15 *wait's*), depending on the block size. The MIN and MAX statements relate to the quantitative algorithms mentioned in Section 3.3. These algorithms are able to determine, respectively, the minimum and maximum distance between two events. In the case of this model the events are the beginning and the end of a cycle of service. Using these algorithms we were able to determine the bounds of the server in terms of the number of users served simultaneously.

```
...
 while (true) {
   ...
   seekTime = select {2,3,4}; /* valid seek time */
   while (seekTime > 0) {
     wait (1); /* passage of seek time */
     seekTime = seekTime - 1;
   }
   readingTime = select {12,13,14,15}; /* valid disk time */
   while (readingTime > 0) {
     wait (1); /* passage disk transfer time */
     readingTime = readingTime - 1;
   }
  ...
 }
 ...

/* minimum and maximum service time in a cycle */
  MIN (beginningOfService, endOfService);
  MAX (beginningOfService, endOfService);
```

Figure 8: Portions of the specification of the ALMADEM-VOD server in Verus.

6 Results

We have compared the quantitative results provided by Verus with the empirical results obtained from the server. The critical system parameters, discussed in Section 5, are the same both for the Verus model and for the version of the server used in our experiments.

In Figure 9, we illustrate how the service time evolves as the number of clients in the system increases. The continuous curve is relative to measurements obtained from the ALMADEM-VOD server, while the dashed curves illustrate the minimum and maximum service times according to Verus. We observe two major facts: (1) the ALMADEM-VOD server always operates at or above the maximum service time predicted by Verus; (2) the ALMADEM-VOD server presents an unexpected non-linearity in its service time when saturation has not been reached yet (i.e., with 25 clients in the system).

These two observations motivated a thorough inspection of the implementation of our server and of

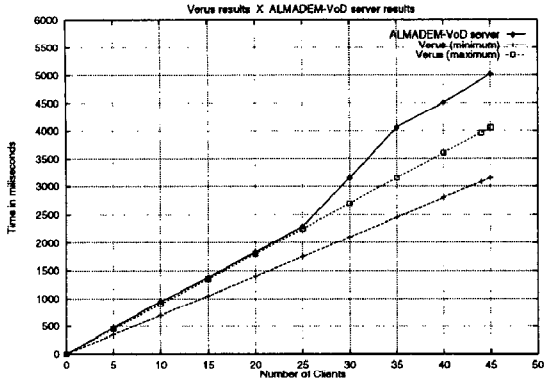

Figure 9: Variation of the service time occupation by clients for the server and for Verus ($T = 4s$).

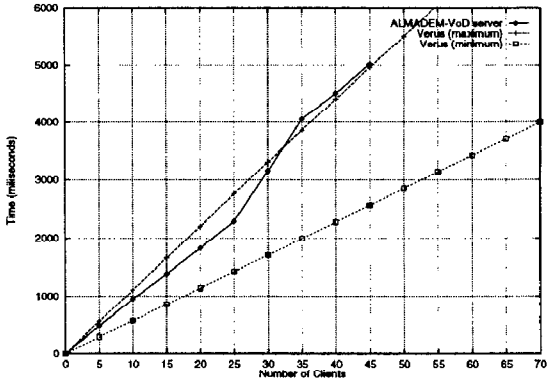

Figure 10: Service time for the ALMADEM-VOD server and for the revised Verus model ($T = 4s$).

```
disk {                              network {
  while (1) {                         while (1) {
    ...                                 ...
    pthread_cond_signal(&cs);           pthread_cond_wait(&cs,&mtx);
    ...                                 ...
    Read blocks from disk;              mini-cycles();
    ...                                 ...
  }                                   }
}                                   }
```

Figure 11: Synchronization of disk and network threads through a pair of *signal* and *wait* primitives.

the Verus model we built, in an attempt to make the results generated by the ALMADEM-VOD server and by Verus consistent.

We have analyzed the dynamic behavior of the Verus model and it was working properly. We have then measured once more the various system parameters we were using and found a problem. The Verus model we built considers a constant disk transfer rate. However, modern disk devices use a multi-zone organization in which there are more sectors per track in the outer tracks. As a result, the outer tracks yield a higher transfer rate because their tangencial speed is higher. To correct the problem, we have changed the Verus model to consider that the transfer rate assumes basically 4 distinct values depending on the track the block of data is in. The new predictions for the service time are shown in Figure 10. We observe that now the maximum service times predicted by the revised Verus model are higher.

However, under heavy load, the service times obtained from the ALMADEM-VOD server continue to exceed the maximum service times predicted by Verus, as illustrated in Figure 10. Further, the non-linearity in the service time of the server is still without explanation. Clearly, the implementation of the server is running into non anticipated overheads. In the search for an explanation, we have investigated the code for the VoD server and then noticed a peculiarity which had not been accounted for in the Verus model we built. This peculiarity is as follows.

In our laboratory, the video server sends blocks of film to its clients through a Myrinet switch which runs at a raw bandwidth of 1 Gbps (giga bits per second). At this bandwidth, conventional implementations of the link layer are unable to handle all the data which arrives at the network physical device. The result, which we observed systematically, is that packets of data are lost at the client machine if a block of film large enough is passed at once to the Linux link layer at the server side. Unfortunately, the block sizes which the server uses (between 500K and 600K bytes) are large enough to cause the problem.

To deal with this problem, we have changed the implementation of the *network* thread to include the notion of mini-cycles. In each mini-cycle, only a portion of each block of film (called a *mini-block*) is sent to each client. While the link layer of a client X handles the reception of a mini-block, mini-blocks can be sent to the other clients in the system. By doing so, we avoid overloading the link layer at client machines. As a result, packet losses are no longer observed.

Mini-cycles are a technical solution to a technological mismatch i.e., current network devices are too fast for conventional operating systems. In this regard, mini-cycles are not really a part of the design of the ALMADEM-VOD server and were not consid-

ered in the Verus model we have built. Since we have tested our implementation of mini-cycles extensively, it seemed to us that mini-cycles would not interfere with the server operation. However, they do. If we simply run the mini-cycles, in situations of light load (for instance, when there is only one client in the system) too many mini-blocks will be sent to each client machine at once, overloading the corresponding network device. To deal with this type of problem, we have decided to put the *network* thread to sleep within a mini-cycle, such that each mini-cycle would have a minimal duration. As a result, the dynamic behavior of the system is now far more complex, because we have to manage several mini-cycles (with service and sleeping times) within each service cycle. At this point, a programming mistake was done.

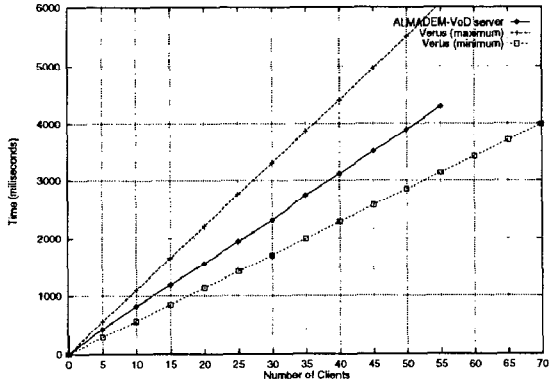

Figure 12: Service time for the new versions of the server and of the Verus model ($T = 4s$).

To simplify the dynamics of the server operation, an attempt was made to guaranttee that the *network* thread would not starve neither be overflown with too much data. To accomplish this effect, the programmer synchronized the *network* and *disk* threads through a pair of *signal* and a *wait* primitives, as illustrated in Figure 11. The idea is that the *network* thread would wait until the *disk* thread indicates that the blocks of film are available in the buffer. This was decided at implementation time as an extra measure to ensure consistent behavior. However, the side-effect is that true concurrency is prevented which results in considerable overhead when the system operates in situations of medium to high load. This overhead led to the results observed in Figure 10.

To fix the problem, we removed the *signal* and *wait* primitives from the code. As a result, the evolution of the service time is now as illustrated in Figure 12. As observed, the service time for the ALMADEM-VOD now increases linearly, as predicted by Verus. Further, it is within the minimum and maximum service times indicated by Verus. Since the period is 4000 mili-seconds (i.e., $T = 4s$), we expect that the server will be able to attend a maximum of 50 clients on average. According to Verus, this maximum will be close to 35 clients in the worst possible situation and close to 70 in the best case scenario.

7 Conclusions

In this work, we have discussed how formal verification can help with the design of multimedia systems in a variety of ways. The approach can be used to check the correctness of the system as well as to assist in determining the system performance parameters and in optimizing its design. We have focused on the application of the Verus tool, which is based on symbolic model checking and has been used to verify a number of real-time applications in the past, to a specific component of a multimedia system — the VoD server.

We have modeled in Verus a low cost PC-based video on demand server called ALMMADEM-VOD. We have then compared the quantitative estimates provided by Verus with empirical measures obtained from the server. Such comparative analysis led us to investigate two main questions. First, why is the service time observed empirically at or above the maximum service time predicted by Verus? Second, why does the service time observed empirically present a non-linearity when Verus predicts that it should increase linearly with the number of clients in the system?

The investigation of these two main questions allowed us to tune the Verus model we have built and also to improve the performance of the ALMADEM-VOD server over 40%. The model was tuned by considering that the disk transfer rate is a function of the position of the track on disk (because modern disk devices use a multi-zone organization in which the transfer rate is higher at the outer tracks). The implementation was improved by removing from the code a *signal-wait* dependency between the disk and network threads, which was mistakenly introduced at programming time. When this dependency was removed, the non-linearity in the evolution of the ser-

vice time disappeared. Finally, our results illustrate how symbolic model checking verification can be of great value in the analysis of the dynamic behavior of a multimedia system.

References

[1] R. Alur, C. Courcourbetis, and D. Dill. Model-checking for real-time systems. In *5th Symposium on Logic in Computer Science*, 1990.

[2] S. Berson, R.Muntz, S. Ghandeharizadeh, and X. Ju. Staggered striping in multimedia information systems. In *ACM SIGMOD Conference*, 1994.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[4] J. Burch, E. Clarke, K. McMillan, D. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Symp. on Logic in Comp. Science*, 1990.

[5] S. Campos. *A Quantitative Approach to the Formal Verification of Real-Time Systems.* PhD thesis, Carnegie Mellon University, 1996.

[6] S. Campos, E. Clarke, W. Marrero, and M. Minea. Timing analysis of industrial real-time systems. In *Workshop on Industrial-strength Formal specification Techniques*, 1995.

[7] S. Campos, E. Clarke, W. Marrero, and M. Minea. Verifying the performance of the PCI local bus using symbolic techniques. In *Intl. Conf. on Computer Design*, 1995.

[8] S. Campos, E. Clarke, W. Marrero, and M. Minea. Verus: a tool for quantitative analysis of finite-state real-time systems. In *ACM Workshop on Languages Compilers and Tools for Real-Time Systems*, 1995.

[9] S. Campos, E. Clarke, W. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *IEEE Real-Time Systems Symp.*, 1994.

[10] S. Campos and O. Grumberg. Selective quantitative analysis and interval model checking: Verifying different facets of a system. In *Conference on Computer-Aided Verification.* Springer-Verlag, 1996.

[11] T. Chua, J. Li, B. Ooi, and K. Tan. Disk striping strategies for large video-on-demand servers. In *ACM Intl. Multimedia Conf.*, pages 297–306, 1996.

[12] E. Clarke and E. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop*, volume 131 of *LNCS.* Springer-Verlag, 1981.

[13] R. Cleaveland, P. Lewis, S. Smolka, and O. Sokolsky. The concurrency factory: a development environment for concurrent systems. In *8th Conf. on Computer-Aided Verification*, volume 1102 of *LNCS.* Springer-Verlag, 1996.

[14] P. Clements, C. Heitmeyer, G. Labaw, and A. Rose. MT: a toolset for specifying and analyzing real-time systems. In *IEEE Real-Time Systems Symp.*, 1993.

[15] J.-P. Courtiat and R. Oliveira. Proving temporal consistency in a new multimedia synchronization model. In *ACM Intl. Multimedia Conf.*, pages 141–152, 1996.

[16] D.A.Dulce, D.J.Duke, G.Faconti, I.Hermam, and M.Massink. Premo: a case study in formal methods and multimedia system specification, 1997. CWI's techinical report: INS-R9708.

[17] A. N. Fredette and R. Cleaveland. RTSL: a language for real-time schedulability analisys. In *IEEE Real-Time Systems Symp.*, 1993.

[18] C.S. Freedman and D.J. DeWitt. The SPIFFI scalable video-on-demand system. In *ACM Intl. Multimedia Conf.*, pages 352–363, 1995.

[19] R. Gerber and I. Lee. A proof system for communicating shared resources. In *IEEE Real-Time Systems Symp.*, 1990.

[20] S. Ghandeharizadeh, S. Kim, and C. Shahabi. On configuring a single disk continuous media server. In *ACM Sigmetrics Performance*, 1995.

[21] T. Henzinger, P. Ho, and W. Wong-Toi. Hytech: the next generation. In *IEEE Real-Time Systems Symp.*, 1995.

[22] J. Lehoczky, L. Sha, J. Strosnider, and H. Tokuda. Fixed priority scheduling theory for hard real-time systems. In Andre M. van Tilborg and Gary M. Koob, editors, *Foundations of Real-Time Computing - Scheduling and Resource Management.* Kluwer, 1991.

[23] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

[24] K. McMillan. *Symbolic Model Checking.* Kluwer, 1993.

[25] B. Ozden, R. Rastogi, and A. Silberschatz. On the design of a low-cost video-on-demand storage system. In *ACM Int. Multimedia Conference*, pages 40–54, 1996.

[26] C. Santos, L. Soares, G. Souza, and JP. Courtiat. Design methodology and formal validation of hypermidia documents. In *ACM Intl. Multimedia Conf.*, pages 39–48, 1998.

[27] J.R. Santos and R. Muntz. Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations. In *ACM Intl. Multimedia Conf.*, 1998.

[28] J. M. Spivey. *Introducing Z: a Specification Language and its Formal Semantics.* Cambridge Press, 1988.

[29] M. Vernick, C. Venkatramani, and T. Chiueh. Adventures in building the stony brook video server. In *ACM Intl. Multimedia Conf.*, pages 287–295, 1996.

[30] J. Yang, A. K. Mok, and F. Wang. Symbolic model checking for event-driven real-time systems. In *IEEE Real-Time Systems Symp.*, 1993.