

Formal Verification and Analysis of Multimedia Systems

Sergio Campos Berthier Ribeiro-Neto Autran Macedo Luciano Bertini

Computer Science Department
Federal University of Minas Gerais
Brazil

Abstract

In this work we discuss the use of formal methods tools, particularly symbolic model checking, in the verification and analysis of multimedia systems. We focus on the use of the Verus tool. Verus is based on symbolic model checking and has been used to verify a number of real-time applications. We show that it can be used not only to check the correctness of a multimedia system, but also to assist in the design of more efficient systems. In this work in particular, we apply Verus to the verification of a low cost video on demand server called ALMADEM-VoD. Modeling this server in Verus provides great insight into its design and its dynamic behavior. Using the quantitative estimates provided by Verus, we check the empirical results generated by our server. Such comparative analysis allows us to identify imperfections in the model and also to detect programming mistakes in the implementation of our server, which would have been difficult to detect otherwise. The correction of such mistakes leads to improved performance and increased reliability.

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 [14, 20]. 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 properties of the system.

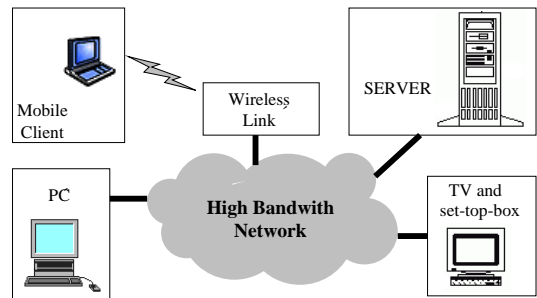


Figure 1: Overall architecture for a video service.

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 cannot provide adequate error coverage, therefore an alternative

approach is the utilization of formal techniques for the verification of the multimedia system.

The approach we adopt here is based on *symbolic model checking* [3, 16], 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* [5, 7] — a software environment which can be used to verify multimedia 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 ALMADEM-VoD, developed within the ALMADEM¹ 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 constraints of the application 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 attended simultaneously).

We model the ALMADEM-VoD server in *Verus* to verify its timing properties. We also analyze quantitative estimates provided by *Verus* for critical system parameters. These estimates are compared with empirical results obtained from the server. This comparative analysis allows us to better understand the operation of the server, a knowledge which can be used to improve its implementation.

Other approaches for verifying the correctness of temporal features of a multimedia application have been proposed in the literature. In [18], the temporal consistency of a hypermedia document is verified using an RT-LOTOS description, which is generated automatically from the document specification. A

¹ALMADEM (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.

similar approach is adopted in [11] which proposes a synchronization model to verify the temporal consistency of a multimedia document. In [12], a suite of formal methods is used to verify the correctness of PREMIO — a standard for the presentation of multimedia 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.

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

2 The *Verus* Modeling Tool

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 seconds [4, 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 [8], and a distributed heterogeneous real-time system [6]. In this work, we use *Verus* to verify the ALMADEM-VoD video server — a low cost VoD server implemented on a PC platform.

2.1 Modeling a Multimedia System

A model of a multimedia system in Verus is a labeled state-transition graph M . The labels correspond to the values of the variables in the program, while the transitions correspond to the passage of time in the model. The key to the efficiency of the algorithms is the use of *binary decision diagrams* (BDDs) to represent the labeled state-transition graph and to verify if a timing property is true or not.

Binary Decision Diagrams

Binary decision diagrams (BDDs) are a canonical representation for Boolean formulas [2]. 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 2 illustrates the BDD for the Boolean formula $(a \wedge b) \vee (c \wedge d)$.

2.2 Symbolic Representation of Transition Graphs

In Verus, the system is represented as a state-transition graph, as illustrated in Figure 3. In this graph, system variables are modelled as atomic propositions, such that each atomic proposition is associated with a Boolean variable created for this purpose. An assignment of values to all the Boolean variables defines a state in the graph (we assume that different

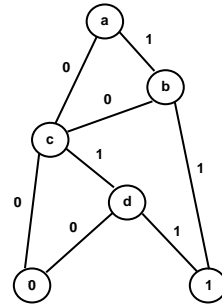


Figure 2: BDD for $(a \wedge b) \vee (c \wedge d)$

states have different labels as described in [16]). For example, if the model has three Boolean propositions a , b , and c , then $(a=1, b=1, c=1)$, $(a=0, b=0, c=1)$, and $(a=1, b=0, c=0)$ are examples of possible states. The *symbolic representations* of these states are (a, b, c) , (\bar{a}, \bar{b}, c) , and (a, \bar{b}, \bar{c}) , respectively, where a means that the variable is true in the state and \bar{a} means that the variable is false. Boolean formulas over variables of the model can be true or false in a given state. The value of a Boolean formula in a state is obtained by substituting into the formula the values of the variables in that state. For example, the formula $a \vee c$ is true in all the three example states discussed above. The graph representation used by our algorithms is a direct consequence of this observation. We use a Boolean formula to denote the set of states in which that formula is satisfied. For example, the formula *true* represents the set of all states, the formula *false* represents the empty set with no states, and the formula $a \vee c$ represents the set of states in which a or c are true. Notice that individual states can be represented by a formula with exactly one proposition for each variable in the system. For instance, the state $s = (a, \bar{b}, c)$ is represented by the formula $a \wedge \bar{b} \wedge c$. We say that $a \wedge \bar{b} \wedge c$ is the formula associated with the state s . Because symbols are used to represent states, algorithms that use this method are called symbolic algorithms.

Transitions can also be represented by Boolean formulas. A transition $s \rightarrow t$ is represented by using two distinct sets of variables, one set for the current state

connectives \neg and \wedge , and *temporal operators*. Each of these operators consists of two parts: a path quantifier followed by a temporal quantifier. Path quantifiers indicate that the property should be true of *all* paths from a given state (**A**), or *some* path from a given state (**E**). The temporal quantifier describes how events should be ordered with respect to time for a path specified by the path quantifier. Examples of temporal quantifiers and their informal meanings are: **F** φ , meaning that φ holds sometime in the future; **G** φ , meaning that φ holds globally on the path; **X** φ , meaning that φ holds in the next state. Some examples of CTL formulas are given below to illustrate the expressiveness of the logic.

- **AG**($req \rightarrow \mathbf{AF} \text{ack}$): It is always the case that if the signal *req* is high, then eventually *ack* will also be high.
- **EF**($started \wedge \neg ready$): It is possible to get to a state where *started* holds but *ready* does not hold.

Real-time CTL, RTCTL, is the extension to CTL that allows the verification of time bounded properties [13]. RTCTL formulas can be obtained by adding time intervals for the satisfaction of each CTL operator. For example, we can specify that an *ack* will arrive shortly after a *req* using the formula **AG**($req \rightarrow \mathbf{AF}_{[0,10]} \text{ack}$). Further details on the semantics of the operators and on the expressiveness of the logic can be obtained in [7].

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 [4, 9].

3 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 [14, 17, 20]. As a result, the server is implemented on a PC-based platform running the Linux operating system. To fulfill 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.

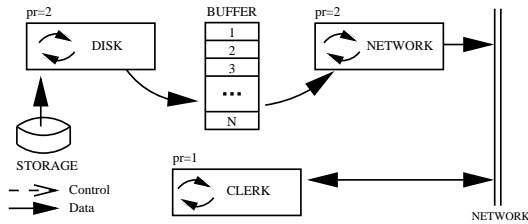


Figure 5: Software architecture of the ALMADEM-VoD video server.

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. 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 implementation, which is adopted in this study, the ALMADEM-VoD server considers a *contiguous* layout of films in disk. In this layout, all blocks of a same film are stored con-

tiguously on disk. More sophisticated layout schemes, involving striping techniques [1, 10], region-based allocation [15], and randomized placement [19], 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 across 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 exhibition. 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. 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 in two fundamental delays:

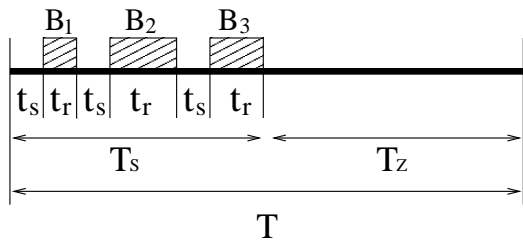


Figure 6: Service cycle with a duration of T seconds.

a seek time t_s to position the disk head at the proper cylinder and a transfer time t_r to read the block of data from disk (there is also a rotational delay which is not shown in Figure 6). 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 across 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 (CTL) block

instead of a Constant Time Length (CTL) block. The length of the blocks in which a given film is divided is determined by the maximum consumption 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 [20].

Let Rc_i be the rate (in bytes per second, or Bps) with which the i th client consumes a block of data. Further, let B_i be the size (in bytes) of the blocks of data for the i th 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 secondary 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 Rd . 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

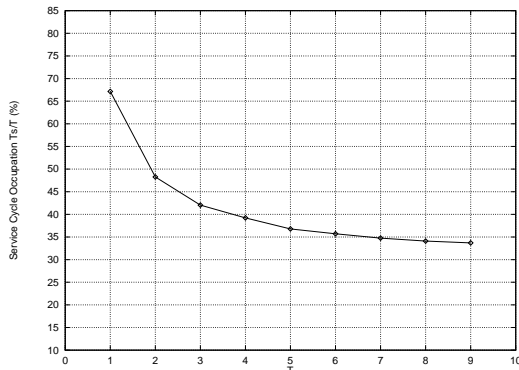


Figure 7: Service cycle occupation in the ALMADEM-VoD server for values of period varying from $T = 1s$ to $T = 9s$. The number of clients in the system is 20.

number of clients which can be served). However, an excessive increase in the service cycle is counter-productive 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 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. As shown, the amortization of the seek time is minimal after $T = 7s$ and does not improve much after $T = 4s$. For periods larger than 4 seconds, we incur in 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.

4 Modeling the ALMADEM-VoD Server in Verus

Verus has a specification language that resembles 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 the writing of a program in C.

To model the behavior of the ALMADEM-VoD server in Verus, we have considered the following system parameters (obtained from the version of the server used in our experiments): (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 7,8 MBps (mega bytes per second); (6) a disk seek time which varies from 10 ms to 20 ms.

The Verus program for the ALMADEM-VoD server has two parts: one called *disk* and another called *spec*. The *disk* routine is an infinite loop that models service cycles of 4s, as done by the *disk* thread in the ALMADEM-VoD server. In a cycle, serving any client requires executing a seek operation and reading a block of data from disk. The programming of these two events in Verus is illustrated in Figure 8. The *wait* command models the passage of time.

The *spec* routine in our Verus program allows the verification of quantitative parameters which reflect the performance of the ALMADEM-VoD server. For instance, given a number of clients, we can ask Verus what are the minimum and maximum possible values for the service time, as illustrated in Figure 8. These

```

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

limits define the best and worst possible time performance for that number of clients.

5 Results

In this section, we verify and analyze the behavior of the ALMADEM-VoD server. This is accomplished by comparing the quantitative results provided by Verus with the empirical results obtained from the server. The critical system parameters, discussed in Section 4, 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

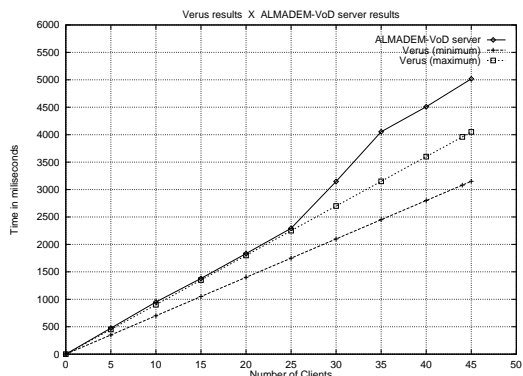


Figure 9: Variation of the service time (with the number of clients in the system) for the ALMADEM-VoD server and for Verus. Period is $T = 4s$.

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 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 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 tangential 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-

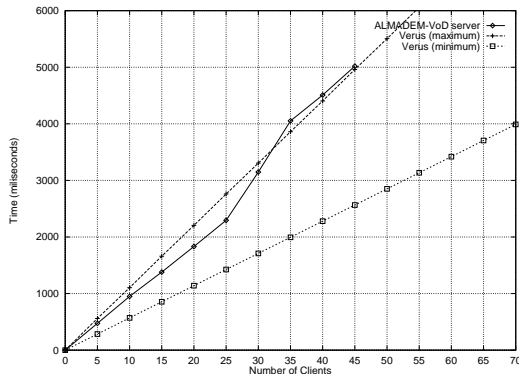


Figure 10: Service time for the ALMADEM-VoD server and for the revised Verus model. Period is $T = 4s$.

linearity in the service time of the ALMADEM-VoD server is still without explanation. Clearly, the implementation of the server is running into overheads which were not anticipated. 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 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

```

disk {
  while (1) {
    ...
    pthread_cond_signal(&cs);
    ...
    Read blocks from disk;
    ...
  }
}

network {
  while (1) {
    ...
    pthread_cond_wait(&cs,&mtx);
    ...
    mini-cycles();
    ...
  }
}

```

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

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 considered 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.

To simplify the dynamics of the server operation, an attempt was made to guarantee that the *network* thread would not starve neither be overflowed 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

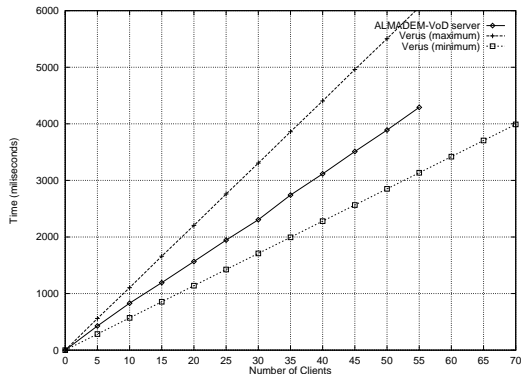


Figure 12: Service time for the new versions of the ALMADEM-VoD server and of the Verus model. Period is $T = 4s$.

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 milliseconds (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.

6 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 ap-

plication 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 modelled in Verus a low cost PC-based video on demand server called ALMADEM-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 implementation of the server. 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 service time disappeared.

Our results illustrate that verification based on symbolic model checking can be of great value in the analysis of the dynamic behavior of a multimedia system. Furthermore, the approach is more effective than simulation studies and can be carried out in shorter time.

References

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

- [2] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.
- [3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Symposium on Logic in Computer Science*, 1990.
- [4] S. Campos, E. Clarke, W. Marrero, and M. Minea. Verifying the performance of the pci local bus using symbolic techniques. In *International Conference on Computer Design*, 1995.
- [5] 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.
- [6] 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.
- [7] S. V. Campos. *A Quantitative Approach to the Formal Verification of Real-Time Systems*. PhD thesis, Carnegie Mellon University, 1996.
- [8] S. V. Campos, E. M. Clarke, W. Marrero, and M. Minea. Timing analysis of industrial real-time systems. In *Workshop on Industrial-strength Formal specification Techniques*, 1995.
- [9] S. V. Campos, E. M. Clarke, W. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *IEEE Real-Time Systems Symposium*, 1994.
- [10] T.S. Chua, J. Li, B.C. Ooi, and K. Tan. Disk striping strategies for large video-on-demand servers. In *ACM Int. Multimedia Conference*, pages 297–306, 1996.
- [11] J.-P. Courtiat and R.C. de Oliveira. Proving temporal consistency in a new multimedia synchronization model. In *ACM Int. Multimedia Conference*, pages 141–152, 1996.
- [12] 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 technical report: INS-R9708.
- [13] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Lecture Notes in Computer Science*, volume 531, pages 136–45. Springer-Verlag, 1990.
- [14] C.S. Freedman and D.J. DeWitt. The spiffi scalable video-on-demand system. In *ACM Int. Multimedia Conference*, pages 352–363, 1995.
- [15] S. Ghandeharizadeh, S.H. Kim, and C. Shahabi. On configuring a single disk continuous media server. In *ACM Sigmetrics Performance*, 1995.
- [16] K. L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.
- [17] 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.
- [18] C.A.S. Santos, L.F.G. Soares, G.L. de Souza, and J.-P. Courtiat. Design methodology and formal validation of hypermedia documents. In *ACM Int. Multimedia Conference*, pages 39–48, 1998.
- [19] J.R. Santos and R. Muntz. Performance analysis of the rio multimedia storage system with heterogeneous disk configurations. In *ACM Int. Multimedia Conference*, 1998.
- [20] M. Vernick, C. Venkatramani, and T. Chiueh. Adventures in building the stony brook video server. In *ACM Int. Multimedia Conference*, pages 287–295, 1996.