To be included in the Proceedings of the IEEE ICMCS'99 Conference.

On Satisfying Timing and Resource Constraints in Distributed Multimedia Systems

Costas Mourlas and David Duce and Michael Wilson
Rutherford Appleton Laboratory, Chilton, Didcot, OX11 0QX, UK
E-mail: {*C.Mourlas,D.A.Duce,M.D.Wilson*}*@rl.ac.uk*

Abstract

In this paper[1]*, we present a new synchronization strategy for multimedia applications executed in a distributed environment. This strategy makes the timing properties of the system and the quality of the media presentations predictable since one is able to determine analytically whether the timing requirements of each multimedia application will be met, and if not, which timing requirements will fail. The proposed synchronization protocol provides deterministic guarantees and service reliability that can't be compromised by resource contention. Thus, the application can maintain the initial quality of service (QoS) level without encountering unpredictable delays and blocking due to synchronization.*

1. Introduction

There is currently considerable interest in developing multi-media applications in open distributed systems. This is motivated by the wide range of potential applications such as desktop conferencing, distributed multi-media information systems and video-on-demand services. However, it is clear that existing frameworks for open distributed systems do not support the particular requirements of distributed multi-media such as real-time constraints, intra/inter-media synchronization and real-time communication. There is also a lack of a suitable theory that makes the timing properties of a distributed multi-media application predictable.

Because of the layered design of multimedia systems, the granularity of synchronization is generally coarser at the application level, becoming more detailed at the lower levels of the system. For example, a user at the application level is concerned that a video segment begins and ends at specific time points whereas the system might be concerned with frame synchronization, real-time frame delivery and resource management. Our work is concentrated on *system-level* synchronization techniques providing a new resource management strategy for multimedia applications executed in a *heterogenous, distributed* environment.

A significant amount of work has been carried out for making resource allocations to satisfy specific application-level requirements and various scheduling schemes are available to ensure that the allocation decisions can be carried out. Various system-wide schemes have been studied to arbitrate resource allocation among contending applications. The Rialto operating system [2] was designed to support simultaneous execution of independent real-time and non-real-time applications, meeting the real-time requirements of all those for which it is possible while providing liveness for the non-real-time programs. The RT-Mach micro-kernel [3] supports a processor reserve abstraction which permits threads to specify their CPU resource requirements. If admitted by the kernel, it guarantees that the requested CPU demand is available to the requestor. Q-RAM [8] and SMART [7] support applications with time constraints, and provides dynamic feedback to applications to allow them to adapt to the current load. The Lancaster QoS Architecture [1] provides extensions to existing micro-kernel environments for the support of continuous media. The QoS Broker model [6] addresses also the requirements for resource guarantees, QoS translation and admission control, so a new system architecture is proposed which provides all these issues. The Nemesis operating system is described in [4] as part of the Pegasus Project, whose goal is to support both traditional and multimedia applications. A large portion also of real-time scheduling theory deals with the important problem of the schedulability analysis and the predictability of a set of real-time applications [9].

We have to notice at this point that most of the above CPU allocation schemes are based on the restrictive assumption that the applications are independent of one another and do not have access to multiple resources simultaneously. In our approach we focus on "real" environments where a set of multimedia applications share a number of non-preemptable resources or access shared data (e.g. storage servers, live media sources etc.) which are part of a high-speed local area network (see figure 1). The proposed synchronization protocol, called the *Set Based Synchronization Protocol*, is

---

[1] This work has been carried out as part of the *ERCIM Fellowship Programme*

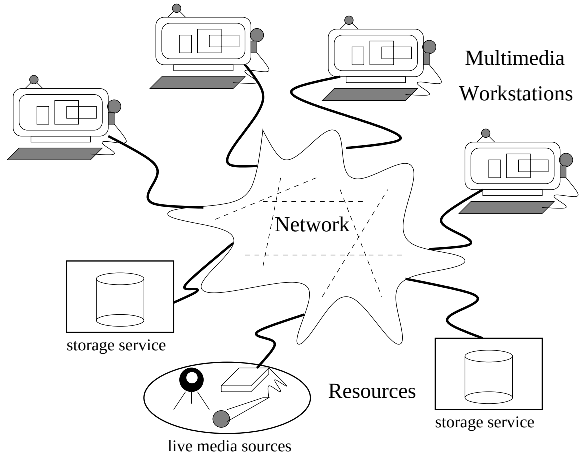

Figure 1. The distributed multimedia environment

based on the on-demand paradigm where resources are assigned only when actually required. The penalty paid for this, found also in all the on-demand approaches, is blocking. In predictable multimedia environments, the blocking has to be deterministic and for this reason our approach imposes a specific structure on blocking to bound the blocking time. The proposed protocol is an extension of our previous work [5] on distributed real-time systems.

2. Assumptions and Notation

In our model, we assume that the QoS for continuous media objects is expressed with temporal and spatial resolutions. The temporal resolution can be expressed by the number of frames per second (fps) or sample rate and the spatial resolution can by expressed by data size, number of bits per pixel, compression scheme, etc. It is assumed also that every application executes periodic reads of a number of frames from a remote media server into a local buffer first and then plays them due to the fact that continuous media require periodic service activities for transmission and presentation.

In this paper, we view every different multimedia application executed in a distributed environment as a *periodic task* that can require in each period the use of non-preemptable resources or access shared data. For example, one multimedia session can be modelled as a task which in every 50 ms needs to deliver 3 video frames from the $storage\_server_1$ and 6 audio frames from $storage\_server_2$. Since these storage servers are shared and exclusively used (i.e. guaranteed exclusive access), there is a possibility for one such task to block waiting for the use of these servers. The period of each multimedia task is determined by the desired quality of service (i.e. the temporal and spatial resolutions of the continuous media), the number of continuous media used in the application, the processor speed and the buffer size used by the node that executes the application. This means that high quality applications using many continuous media are represented by our model as tasks having short periods, i.e. high frequency tasks.

The assumptions and basic notation that will be used throughout this paper follows:

1. any continuous media application is represented by a multimedia task τ_i allocated on a different node $\wp_i$ of the distributed system and can require the use of non-preemptable resources or access shared data R_i.

2. every multimedia task τ_i is periodic with period T_i and has deadline D_i at the end of its period (i.e. $D_i = T_i$).

3. multimedia tasks are assigned fixed priorities inversely to their periods.

4. every task asks for all of its global resources R_i only once in its period and subsequently can release these resources. Two operations are used for this reason:
   – *allocate(ResourceSet)* and
   – *release(ResourceSet)*.
   When a task τ_i issues the $allocate$ command asking for its resources it then blocks (i.e. hangs) until all these resources have been allocated to τ_i by the resource manager. The duration of this time interval constitutes the blocking time B_i of the task. The section between exiting from the $allocate$ call and the last $release$ call constitutes the *critical section* of the task.

5. every multimedia task τ_i has known, deterministic worst-case execution time C_i. This is the total deterministic computation requirement of task τ_i during each period, and $C_i = C^i_{cs} + C^i_{non-cs}$ where:

   C^i_{cs} is the computation requirement of task τ_i within its critical section. This is the total time that τ_i uses the resources and the network in each period for data retrieval,

   C^i_{non-cs} is the deterministic computation requirement of task τ_i outside its critical section. This is the time in each period that τ_i needs to process the received data frames.

Due to the fact that every multimedia task is allocated on a different node of the distributed system, cpu scheduling is not the main problem, but since tasks are inter-dependent the main problem is task synchronization and resource allocation. Hence, blocking due to synchronization has to be deterministic in order to have nice analysis properties and a high degree of system predictability.

3. The Set Based Synchronization Protocol

In this section, we present the Set Based Synchronization Protocol suitable for synchronizing multimedia tasks executing on distributed systems. A set of n multimedia applications can be modelled as a set of n periodic tasks $\tau_1, \ldots, \tau_n$ each one bound to a different node of the distributed system. Each task is characterized by five components (C^i_{cs}, $C^i_{n\,on-cs}$, T_i, D_i, R_i), $1 \leq i \leq n$, according to the notation and the assumptions introduced in the previous section.

In the analysis of the Set Based Synchronization Protocol, each one of the resources can be in one of the three following different states at a specific point in time during execution:

1. $free$ if there is not any task that either asks for the use of this resource or uses this resource at this time in its critical section.

2. $in\ use$ if there is a task that asked for the use of this resource for its critical section and this task is now within its critical section.

3. $allocated$ (to a task τ_j) if the task τ_j asked for the use of this resource, the resource has been allocated to the task τ_j but τ_j hasn't entered yet into its critical section.

Suppose that a task τ_i requires the use of κ resources through a call of the form $allocate(R_i)$, where $R_i = \{r_1, \ldots, r_\kappa\}$. Then the following cases can occur:

1. All the required resources in R_i are $free$. Then all these resources are $allocated$ to the task τ_i and the task proceeds immediately to its critical section. The states of all resources in R_i become $in\ use$.

2. If case 1 does not hold then the following actions are performed. If any of the $r_1, \ldots, r_\kappa$ is $free$ then it is $allocated$ to τ_i. If any of the resources in R_i has been $allocated$ to a lower priority task τ_j ($T_i < T_j$) and τ_j has not entered its critical section then it is deallocated from τ_j and it is $allocated$ to τ_i. If any of the resources in R_i is $in\ use$ then after its release it is $allocated$ to the highest priority task that is requesting it. The task τ_i will proceed to its critical section if and only if all the resources in R_i have been $allocated$ to τ_i.

We have to notice here that a resource is actually locked by a task only if the resource is $in\ use$ by this task and not when the resource is $allocated$ to that task. Note also that by the definition of the protocol, a task τ_i can be blocked by a lower priority task τ_j, only if τ_j is executing within its critical section when τ_i requested resources and both τ_i and τ_j use common resources in their critical section, i.e. $R_i \cap R_j \neq \emptyset$.

Theorem 3.1 The Set Based Synchronization Protocol prevents deadlocks.

Proof: By definition, every task τ_i proceeds to its critical section if and only if all the resources in R_i have been $allocated$ to τ_i. Thus, τ_i will never ask in its critical section for the use of any other resource and so a blocking cycle (deadlock) cannot be formed. $\Box$

To perform a schedulability analysis using the proposed synchronization protocol, we define $B_\kappa,\ 1 \leq \kappa \leq n$, the longest duration of blocking that can be experienced by τ_κ. Since each task is bound to a different processor, theorem 3.2 defines a sufficient set of conditions for a set of tasks to meet their deadlines.

Theorem 3.2 A set of n periodic tasks, each one bound to a different processor $\wp$ can be scheduled using the Set Based Synchronization Protocol if the following conditions are satisfied:

$$\forall i, 1 \leq i \leq n, \quad C_i + B_i \leq T_i \tag{1}$$

Proof: The above set of inequalities state that for each task τ_i the sum of the blocking time B_i and the total execution time C_i of the task must be lower than or equal to its period T_i. If this sum which represents the completion time of τ_i was greater than its period and hence greater than its deadline (since $T_i = D_i$), task τ_i could not be scheduled. $\Box$

Once B_is have been computed for all i, theorem 3.2 can then be used to determine the schedulability of the set of tasks.

4. Determination of Task Blocking Time

Here, we shall compute the worst-case blocking time that a task has to wait for its resource requirements. The fundamental objective of the Set Based Synchronization Protocol is to obtain bounded blocking times for multimedia tasks requiring the access of shared resources. The bounded waiting times in turn can be used to determine whether a set of multimedia tasks running on a distributed environment can meet their deadlines using theorem 3.2.

Assume a set of n periodic tasks with $D_i = T_i$ using the Set Based Synchronization Protocol. We define as $B_i,\ 1 \leq i \leq n$, the longest duration of blocking that can be experienced by τ_i.

Theorem 4.1 Consider a set of n tasks $\tau_1, \ldots, \tau_n$ each one bound to a different processor $\wp_i$ and the Set Based Synchronization Protocol is used for the allocation of the resources. Let

$H_i \;=\; \{\tau_j \,|\, T_i > T_j \wedge R_i \cap R_j \neq \emptyset\}$, - set of tasks having higher priorities and need common resources with τ_i

$L_i \;=\; \{\tau_j \,|\, T_i < T_j \wedge R_i \cap R_j \neq \emptyset\}$, - set of tasks having lower priorities and need common resources with τ_i

$\beta_i \;=\; \max(\{C^j_{cs} \mid \tau_j \in L_i\})$. - blocking time due to lower priority tasks in L_i

Then, the worst case blocking time B_i of task τ_i is equal to

$$B_i = \begin{cases} \beta_i + \sum\limits_{\forall \tau_j \in H_i} \alpha^i_j & \text{if } \sum\limits_{\forall \tau_j \in H_i} \alpha^i_j < min(\{T_\kappa \mid \tau_\kappa \in H_i\}) \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

where α^i_j is the contribution of task τ_j to the B_i value and,

$$\alpha^i_j = \begin{cases} 0 & \text{if } \exists \tau_l \in H_i : R_j \cap R_l \neq \emptyset \wedge T_j < T_l \\ B_j + C^j_{cs} & \text{otherwise} \end{cases} \tag{3}$$

Proof: The sum in formula 2 above represents the longest blocking time for a task τ_i at its worst-case task set phasing. At this worst-case phasing of the tasks, when τ_i wants to enter into its critical section, it finds the lower priority tasks that use common resources with τ_i executing within their critical sections. Then, just before all these lower priority tasks have finished their critical sections and have released their resources, all the higher priority tasks that use common resources with τ_i come one after the other and ask for their resources, enter their critical sections and at the end release their resources.

Thus, B_i has two parts. The first part, namely β_i is due to L_i and it is equal to the maximum value of C_{cs} among the lower priority tasks in L_i. The second part comes from tasks in H_i. In all cases, the duration of the second part should be less than the minimum period T_κ of the tasks in H_i, otherwise the task τ_κ could block repeatedly the task τ_i and in this case B_i can be prohibitively large or even unbounded (condition of formula 2). This does not mean that the task set is not schedulable, rather that there is not a way to determine accurately using the proposed protocol the worst case blocking time B_i of the task τ_i.

Note now that the longest blocking time B_i occurs when the blocking time of τ_λ, $\tau_\lambda \in H_i$ does not overlap with those of other tasks in H_i at the worst-case task set phasing for τ_i. Thus, a task τ_λ in H_i contributes to B_i by $(B_\lambda + C^\lambda_{cs})$ in case this task does not have common resources with any other task in H_i. Assume now that a task τ_j in H_i has common resources with a task τ_l in H_i and τ_l has lower priority than τ_j. In this case, the contribution of τ_j to the blocking time of τ_i is taken care by its lower priority task τ_l. Thus, τ_j may contribute indirectly to B_i through the worst case blocking time B_l of τ_l. Its direct contribution is zero. As the same argument may equally apply to τ_l, it follows that for any task τ_j in H_i with common resources with others in H_i contributes only by a factor $(B_j + C^j_{cs})$ in case there is not any task τ_k with priority lower than τ_j sharing common resources with τ_j. This condition is expressed by the formula 3. Hence the Theorem follows. □

As far as the case of tasks with equal periods is concerned, it is not necessary to link the priority of a task directly with its period. We can assign to each task τ_i a unique priority p_i such that $\forall i, j \;\; p_i < p_j \Rightarrow T_i \geq T_j$. Once these blocking terms B_i, $1 \leq i \leq n$, have been determined, theorem 3.2 gives a fairly complete solution for multimedia task synchronization and scheduling in the distributed environment.

5. A Schedulability Analysis Example

We illustrate the schedulability analysis based on the proposed synchronization protocol with the following example.

Example 5.1 Consider a distributed environment and four video presentations each one displayed on a different node of the system and represented by a task $\tau_i, 1 \leq i \leq 4$. This environment also supports five storage servers $r_1, \ldots, r_5$ where a collection of continuous media clips is stored.

We allocate to each task τ_i the minimum bandwidth $Bandwidth_i$ that can be provided at the worst case task phasing. We assume also that a circular buffer of size $2{*}Buffer\_size_i$ is reserved in the buffer cache of node $\wp_i$. In each period, while the presentation is consuming $Buffer\_size_i$ bits of data from its buffer, the other $Buffer\_size_i$ bits that the presentation will consume in the next period are retrieved from the storage servers into the buffer. This ensures that each presentation will have sufficient data to display the corresponding streams continuously. In addition, we define the display rate for each task τ_i (the rate at which data are consumed from the buffer for presentation purposes), using the equation:

$$display\_rate_i = spatial\_resolution_i * temporal\_resolution_i.$$

Then, we require the following equalities to hold:

$$C^i_{cs} * Bandwidth_i = Buffer\_size_i \tag{4}$$

$$display\_rate_i * T_i = Buffer\_size_i \tag{5}$$

Equation 4 is used to evaluate the deterministic computation requirements within the critical sections of the tasks. Equation 5 states that with known display rate and buffer size for each task then the duration of consuming all the available data from the local buffer can be evaluated. This time duration represents the period T_i of the task τ_i. The value $C^i_{non\_cs}$ is the computation requirement of task τ_i to process the received data frames of size $Buffer\_size_i$.

In our example, the expected quality of every presentation and the resource requirements R of every presentation

Table 1. Parameters of task set in example

Parameters of Task Set						
Task	Spat. Res.	fps	R	T	$C_{\text{non-cs}}$	C_{cs}
τ_1	160×120×8	22	r_3, r_1	12	3	2
τ_2	140×110×8	20	r_3, r_2	14	3	2
τ_3	130×110×8	19	r_1, r_4	25	4	3
τ_4	120×100×8	15	r_3, r_5	31	6	3

of each task are listed in Table 1. The computed values of the periods and computation times within and outside critical sections for each task are listed in Table 1 in the corresponding columns. The worst-case blocking duration of each task is the sum given in formula 2 and it is determined as follows:

- Task τ_1:
  – $H_1 = \emptyset$ – $L_1 = \{\tau_2, \tau_3, \tau_4\}$
  – $\beta_1 = \max(C^2_{cs}, C^3_{cs}, C^4_{cs}) = 3$
  – $B_1 = \beta_1 + \sum_{\forall \tau_i \in H_1} \alpha^1_i = \beta_1 = 3$ since $H_1 = \emptyset$
- Task τ_2:
  – $H_2 = \{\tau_1\}$ – $L_2 = \{\tau_4\}$ – $\beta_2 = C^4_{cs} = 3$
  – $B_2 = \beta_2 + \sum_{\forall \tau_i \in H_2} \alpha^2_i = \beta_2 + \alpha^2_1 = \beta_2 + (B_1 + C^1_{cs}) = 3 + (3 + 2) = 8$ and the condition $(3+2) < 12$ holds (where 12 is the period of $\tau_1 \in H_2$).
- Task τ_3:
  – $H_3 = \{\tau_1\}$ – $L_3 = \emptyset$ – $\beta_3 = 0$
  – $B_3 = \beta_3 + \sum_{\forall \tau_i \in H_3} \alpha^3_i = \beta_3 + \alpha^3_1 = \beta_3 + (B_1 + C^1_{cs}) = 0 + (3 + 2) = 5$.
- Task τ_4:
  – $H_4 = \{\tau_1, \tau_2\}$ – $L_4 = \emptyset$ – $\beta_4 = 0$
  – $B_4 = \beta_4 + \sum_{\forall \tau_i \in H_4} \alpha^4_i = \beta_4 + (\alpha^4_1 + \alpha^4_2) = \beta_4 + (0 + (B_2 + C^2_{cs})) = 0 + (0 + (8 + 3)) = 11$ since $11 < 12$.

  We have to notice here that the term α^4_1 is zero according to formula 3 since there exists a task, τ_2, where $\tau_2 \in H_4$, $R_1 \cap R_2 \neq \emptyset$ and $T_1 < T_2$. This means that τ_1 does not contribute directly to the total value of B_4 but only indirectly since the blocking time B_2 of task τ_2 includes the occurence of task τ_1.

Computing now the criterion given in Theorem 3.2 we have:

- i=1 $C_1 + B_1 = 5 + 3 = 8 \leq 12 \quad (T_1 = 12)$
- i=2 $C_2 + B_2 = 5 + 8 = 13 \leq 14 \quad (T_2 = 14)$
- i=3 $C_3 + B_3 = 7 + 5 = 12 \leq 25 \quad (T_3 = 25)$
- i=4 $C_4 + B_4 = 9 + 11 = 20 \leq 31 \quad (T_4 = 31)$

We therefore determine that the above set of tasks $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ is schedulable.

6. Conclusions

In this paper, we have presented a scheduling mechanism with an integrated resource allocation model that is analyzable and understandable at a high level. Given a set of multimedia applications we know in advance if they will meet their deadlines or not. This scheduling strategy has been designed for multimedia applications that operate in a distributed computing environment where every multimedia task is allocated on a different node and can require the use of global resources. It is an approach for deterministic guarantees and provides $predictable$ distributed multimedia applications.

It is also important to note that the proposed strategy can be easily implemented especially on a distributed system since only a message passing scheme is required to be supported by the underlying software.

References

[1] G.Coulson, G. Blair, P. Robin, and D. Shepherd. Supporting Continuous Media Applications in a Micro-Kernel Environment. In O. Spaniol, editor, *Architectures and Protocols for High-Speed Networks*. Kluwer Academic Publishers, 1994.

[2] M. B. Jones, D. Rosu, and M. Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, October 1997.

[3] C. Lee, R. Rajkumar, and C. Mercer. Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach. In *Proceedings of the Multimedia Japan 96*, April 1996.

[4] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1280–1297, September 1996.

[5] C. Mourlas and C. Halatsis. Task Synchronization for Distributed Real-Time Applications. In *Proceedings of the Ninth Euromicro Workshop on Real-Time Systems*, pages 184–190. IEEE Computer Society, 1997.

[6] K. Nahrstedt and J. M. Smith. The QoS Broker. *IEEE Multimedia*, 2(1):53–67, Spring 1995.

[7] J. Nieh and M. S. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, October 1997.

[8] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A Resource Allocation Model for QoS Management. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1997.

[9] L. Sha, M. Klein, and J. Goodenough. Rate Monotonic Analysis for Real-Time Systems. In A. van Tilborg and G. Koob, editors, *Foundations of Real-Time Computing: Scheduling and Resource Management*, chapter 5, pages 129–155. Kluwer Academic Publishers, 1991.