



Distributed Systems: Concepts and Design

Edition 3

By George Coulouris, Jean Dollimore and Tim Kindberg
Addison-Wesley, ©Pearson Education 2001.

Chapter 15 Solutions

- 15.1 Outline a system to support a distributed music rehearsal facility. Suggest suitable QoS requirements and a hardware and software configuration that might be used.

15.1 Ans..

This is a particularly demanding interactive distributed multimedia application. Konstantas *et al.* [1997] report that a round-trip delay of less than 50 ms is required for it. Clearly, video and sound should be tightly synchronized so that the musicians can use visual cues as well as audio ones. Bandwidths should be suitable for the cameras and audio inputs used, e.g. 1.5 Mbps for video streams and 44 kbps for audio streams. Loss rates should be low, but not necessarily zero.

The QoS requirements are much stricter than for conventional videoconferencing – music performance is impossible without strict synchronization. A software environment that includes QoS management with resource contracts is required. The operating systems and networks used should provide QoS guarantees. Few general-purpose OS's provide them at present. Dedicated real-time OS's are available but they are difficult to use for high-level application development.

Current technologies that should be suitable:

- ATM network.
- PC's with hardware for MPEG or MJPEG compression.
- Real-time OS with support for high-level software development, e.g. in CORBA or Java.

-
- 15.2 The Internet does not currently offer any resource reservation or quality of service management facilities. How do the existing Internet-based audio and video streaming applications achieve acceptable quality? What limitations do the solutions they adopt place on multimedia applications?

15.2 Ans..

There are two types of Internet-based applications:

- a) Media delivery systems such as music streaming, Internet radio and TV applications.
- b) Interactive applications such as Internet phone and video conferencing (NetMeeting, CuSeemMe).

For type (a), the main technique used is *traffic shaping*, and more specifically, buffering at the destination. Typically, the data is played out some 5–10 seconds after its delivery at the destination. This masks the uneven latency and delivery rate (jitter) of Internet protocols and masks the delays incurred in the network and transport layers of the Internet due to store-and-forward transmission and TCP's reliability mechanisms.

For type (b), the round trip delay must be kept below 100 ms so the above technique is ruled out. Instead, *stream adaptation* is used. Specifically, video is transmitted with high levels of compression and reduced frame rates. Audio requires less adaptation. UDP is generally used.

Overall, type (a) systems work reasonably well for audio and low-resolution video only. For type (b) the results are usually unsatisfactory unless the network routes and operating system priorities are explicitly managed.

-
- 15.3 Explain the distinctions between the three forms of synchronization (synchronous distributed state, media synchronization and external synchronization) that may be required in distributed multimedia applications. Suggest mechanisms by which each of them could be achieved, for example in a videoconferencing application.

15.3 Ans..

synchronous distributed state: All users should see the same application state. For example, the results of operation on controls for a video, such as start and pause should be synchronized, so that all users see the same frame. This can be done by associating the current state (sample number) of the active multimedia streams with each state-change message. This constitutes a form of logical vector timestamp.

media synchronization: Certain streams are closely coupled. E.g. the audio that accompanies a video stream. They should be synchronised using timestamps embedded in the media data.

external synchronization: This is really an instance of synchronous distributed state. The messages that update shared whiteboards and other shared objects should carry vector timestamps giving the states of media streams.

-
- 15.4 Outline the design of a QoS manager to enable desktop computers connected by an ATM network to support several concurrent multimedia applications. Define an API for your QoS manager, giving the main operations with their parameters and results.

15.4 Ans..

Each multimedia application requires a resource contract for each of its multimedia streams. Whenever a new stream is to be started, a request is made to the QoS manager specifying CPU resources, memory and network connections with their Flow Specs. The QoS manager performs an analysis similar to Figure 15.6 for each end-to-end stream. If several streams are required for a single application, there is a danger of deadlock – resources are allocated for some of the streams, but the needs of the remaining streams cannot be satisfied. When this happens, the QoS negotiation should abort and restart, but if the application is already running, this is impractical, so a negotiation takes place to reduce the resources of existing streams.

API:

QoSManager.QoSRequest(FlowID, FlowSpec) -> ResourceContract

The above is the basic interface to the QoS Manager. It reserves resources as specified in the *FlowSpec* and returns a corresponding *ResourceContract*.

A *FlowSpec* is a multi-valued object, similar to Figure 15.8.

A *ResourceContract* is a token that can be submitted to each of the resource handlers (CPU scheduler, memory manager, network driver, etc.).

Application.ScaleDownCallback(FlowID, FlowSpec) -> AcceptReject

The above is a callback from the QoS Manager to an application, requesting a change in the *FlowSpec* for a stream. The application can return a value indicating acceptance or rejection.

-
- 15.5 In order to specify the resource requirements of software components that process multimedia data, we need estimates of their processing loads. How should this information be obtained?

15.5 Ans..

The main issue is how to measure or otherwise evaluate the resource requirements (CPU, memory, network bandwidth, disk bandwidth) of the components that handle multimedia streams without a lot of manual testing. A test framework is required that will evaluate the resource utilization of a running component. But there is also a need for resource requirement models of the components – so that the requirements can be extrapolated to different application contexts and stream characteristics and different hardware environments (hardware performance parameters).

-
- 15.6 How does the Tiger system cope with a large number of clients all requesting the same movie at random times?

15.6 Ans.

If they arrive within a few seconds of each other, then they can be placed sufficiently close together in the schedule to take advantage of caching in the cubs, so a single disk access for a block can service several clients. If they are more widely spaced, then they are placed independently (in an empty slot near the disk holding the first block of the movie at the time each request is received). There will be no conflict for resources because different blocks of the movie are stored on different disks and cubs.

-
- 15.7 The Tiger schedule is potentially a large data structure that changes frequently, but each cub needs an up-to-date representation of the portions it is currently handling. Suggest a mechanism for the distribution of the schedule to the cubs.

15.7 Ans.

In the first implementation of Tiger the controller computer was responsible for maintaining an up-to-date version of the schedule and replicating it to all of the cubs. This does not scale well – the processing and communication loads at the controller grow linearly with the number of clients – and is likely to limit the scale of the service that Tiger can support. In a later implementation, the cubs were made collectively responsible for maintaining the schedule. Each cub holds a fragment of the schedule – just those slots that it will be playing processing in the near future. When slots have been processed they are updated to show the current viewer state and then they are passed to the next ‘downstream’ cub. Cubs retain some extra fragments for fault-tolerance purposes.

When the controller needs to modify the schedule – to delete or suspend an existing entry or to insert a viewer into an empty slot – it sends a request to the cub that is currently responsible for the relevant fragment of the schedule to make the update. The cub then uses the updated schedule fragment to fulfil its responsibilities and passes it to the next downstream cub.

-
- 15.8 When Tiger is operating with a failed disk or cub, secondary data blocks are used in place of missing primaries. Secondary blocks are n times smaller than primaries (where n is the decluster factor), how does the system accommodate this variability in block size?

15.8 Ans.

Whether they are large primary or smaller secondary blocks, they are always identified by a play sequence number. The cubs simply deliver the blocks to the clients in order via the ATM network. It is the clients’ responsibility to assemble them in the correct sequence and then to extract the frames from the incoming sequence of blocks and play the frames according to the play schedule.