Distributed Systems: Concepts and Design

Edition 5

By George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair

Chapter 20 Solutions (Multimedia)

20.1 Give the outline of a system which would support a distributed video conference facility. Suggest suitable QoS requirements and hardware and software configurations that might be used.

20.1 Ans.

Low-latency communication, synchronous distributed state, media synchronization, external synchronization

---

20.2 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?

20.2 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.

20.3 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 video conferencing application.

20.3 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.

20.4 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.

20.4 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 17.6 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 17.8.

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.

20.5 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 without undue effort?

20.5 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).

20.6 When a file is created in BitTorrent, which file is created to hold metadata associated with that file? What information does it hold?

20.6 Ans.

When a file is created in BitTorrent, a *.torrent* file is created which holds meta-data associated with that file. This information includes:

- the name and length of the file,
- the location of a *tracker* (specified as a URL), which is a centralised server that manages downloads of that particular file,
- a *checksum* associated with each chunk, and generated using the SHA1 hashing algorithm, that enables content to be verified following download.

20.7 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.

20.7 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.

20.8 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?

20.8 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.

20.9 Discuss the relative merits of the *rarest-first* download policy in BitTorrent in comparison to the more traditional sequential download approach.

20.9 Ans.

BitTorrent schedules downloads according to a rarest first policy, that is each peer maintains a record of the number of copies of each piece in its peer set (the set of connected peers) and, based on this data structure, schedules downloads by randomly selecting a piece from its rarest piece set. This ensures that rare items (including new items) spread quickly and hence this significantly increases the availability of pieces of data. The sequential download policy has no knowledge of availability of data and hence is not tailored towards this particular outcome.

20.10 Discuss the relative merits of the end-system approaches to video streaming in comparison to video streaming using IP multicast.

20.10 Ans.

Most systems now advocate *end system approaches* to video streaming where control and intelligence resides at the edges of the network and not in the network itself. This approach is also referred to as *application level multicast* and implies the formation of an overlay network to support the associated multimedia traffic.