Resource Management for Real Time Parallel Processing in a Distributed System

MATEI DOBRESCU, STEFAN MOCANU
Faculty of Control and Computers
Politehnica University of Bucharest
Splaiul Independentei 313, Bucharest
ROMANIA

Abstract: The main goal of this work is to develop a computational framework in the context of the bandwidth needs for traffic generated in a distributed multimedia environment. For this purpose a specific platform architecture was proposed, that uses for communications between members real-time dependable (RTD) channels, a communication-oriented abstraction that can be configured to meet the QoS requirements of a variety of distributed applications. This customization ability is based on the construction of middleware services out of software modules called micro-protocols. On the platform prototype was tested a resource allocation protocol that uses a combination of bandwidth reservation and bandwidth borrowing to provide network users with QoS in terms of guaranteed bandwidth, call blocking, and call dropping probabilities. The goal is to make possible wireless connections in the system. Some preliminary simulation results are also presented.

Key-Words: realtime, group multicast, bandwidth allocation, QoS provisioning, multimedia traffic, wireless network, resource management

1 Problem Statement

The system we are developing is essentially a higher-level Computer Supported Collaborative Work (CSCW) system. The framework is a Distributed Virtual Environment (DVE) where multiple users connected by a network can share information with each other. That should be the character of some parallel image processing algorithms, conceived to solve real-time problems connected with 2D or 3D image analysis. In order to ensure also the possibility to have mobile users in the system, we have proposed a connection on Real-Time Dependable (RTD) channels (for the beginning a wireline connection, but with the aim to test wireless connections in the future). Because each connection may have different requirements, providing QoS guarantees poses a difficult challenge. Admission control and bandwidth allocation schemes can help provide these guarantees in wireline networks, but, in wireless networks, the problem is much more complex due to bandwidth limitations and host mobility [1]. The connection or call blocking probability (CBP) is an important QoS parameter in wireless networks. The connection or call dropping probability (CDP) is the likelihood that the network will deny resources to a connection after it is in progress. A connection may be dropped during a handoff, when the host moves from a cell with ample resources into a cell that is too congested to support more traffic. Consequently, minimizing the CDP is one of the main goals of QoS provisioning in wireless networks [2], [3]. It was noticed that multimedia applications can tolerate and gracefully adapt to transient fluctuations in the QoS they receive from the network [4], [5]. In our approach we propose the use of an adjustable-rate codec, along with appropriate buffering before play-out, that can allow applications to adapt to temporary bandwidth fluctuations with little or no perceived degradation in quality.

2 Characteristics of Groupware Systems

Groupware systems are computer-based systems that support two or more users engaged in a common task, and that provide an interface to a shared environment. The main characteristics of group work are:

- Several users physically dispersed need to work on a shared environment.

- Access control mechanisms should be available over several information types handled in the shared environment.
- Suitable communication channels and tools should be provided to the users in order to coordinate activities.

The major characteristic of collaborative systems, absent in traditional applications, is the interaction among several users enabling group activities. There are many potential applications specially indicated to work above collaborative platforms. One example of such activities is the parallel image processing, especially for 3D editing. In the following we will designate this application as PIP (Parallel Image Processing)

Two solutions exist for the development of groupware systems.

- One of them is to integrate the existing single user applications in a distributed platform, with minor changes to the application. In this case, there must be mechanisms available in the layer below the application, in charge of all the operations needed in a groupware session: data distribution, coordination among users and other session control tasks.
- The alternative is to develop groupware applications from the beginning just for use in very specific shared environments. These applications are always collaboration aware, and normally, they provide for specific group interfaces, specially designed for group work.

There are several advantages on the use of the first approach. All the experience and work done with single-user applications can still be used in the new environment. In addition, due to the modular design of the first approach, it is easier to integrate additional functionality using the same platform. Usually, the user interface of such an application is not designed for group interaction, and therefore, additional interface mechanisms should be provided to accomplish this requirement. That's the reason for choosing for our PIP platform the first approach. The highest requirement to the PIP system is the real time cooperative working section. The most related design issues include:

- Admission and withdrawal of members.
- Inclusion or exclusion of distributed applications during the same collaborative session.
- Flow control between the applications and the communication support.
- Registration of specific characteristics for each application involved.
- Exception handling and failure recovery.
- Application consistency
- Consistency and concurrency control.

3 Real-Time Dependable Channels: Customizing QoS Attributes for Distributed Systems

An increasing number of distributed applications must provide enhanced Quality of Service (QoS) guarantees related to dependability and real time. Implementing these requirements individually can be difficult enough, but having to deal with the combination can make the problem virtually intractable. One problem is that the two types of requirements often conflict, resulting in a trade-off situation. For example, increasing the reliability of a communication service through retransmissions may cause a delay that results in a missed deadline. We decided to use for the communication infrastructure real-time dependable (RTD) channels. A RTD channel is a communication-oriented abstraction that is designed to simplify the development of distributed applications with both dependability and real-time requirements. RTD channels are implemented as middleware mechanisms between the operating system and application, thereby providing a virtual machine with enhanced QoS guarantees on which applications can be built. The key characteristic of RTD channels is that they are highly configurable and customizable at a fine-grain level. This customization can be done both at system build time and at channel creation time; build-time customization defines the QoS properties that can be enforced, while open-time customization defines what QoS requirements will be enforced for a given connection. This dual level of customization provides a wide range of flexibility and allows the same supporting software to be used for a variety of applications and for a variety of execution environments.

A real-time dependable (RTD) channel has a combination of dependability and timeliness guarantees. The RTD channel service offers a simple API consisting of operations to open a channel, push a message into a channel, and close a channel. Messages are delivered to the receiver using an upcall

to a specified function. The API is the same regardless of the chosen channel properties.

Channel Shapes

Different types of RTD channels can be defined based on whether the traffic is unidirectional (UD) or bidirectional (BD) and how many processes the channel connects and in what manner. A point-to-point channel (PP) connects two processes, a multitarget (MT) channel connects one source to many targets, and a multisource channel (MS) connects multiple sources to a single target. MT and MS channels are equivalent if the channels are bidirectional, so we use BD/M to refer to either. Finally, a multisource, multitarget channel (MST) connects multiple sources to multiple targets.

Channel Properties

A large number of properties can be defined for channels, but for brevity, we describe only a representative set here. In particular, we consider *real-time* – i.e., whether each message sent on a channel will be delivered to its destinations by a deadline, *reliability* – i.e., whether each message reaches its destinations, *message ordering*–i.e., in what order messages are delivered to the application, and *jitter* – i.e., the variance in message transmission time. Other properties not considered here include atomicity, stability [6], security, and properties related to changes in the set of processes using the channel, such as virtual synchrony [7] and safety.

Event-driven model.

The basic approach of our framework is based on implementing different semantic properties and functional components of a service as separate modules called micro-protocols that interact using an event-driven execution model. A custom version of a service is constructed at build time by choosing a set of micro-protocols and linking them together with the runtime system to form a composite protocol. Once created, a composite protocol is composed in a traditional hierarchical manner with other protocols to form the application's protocol graph. A micro-protocol is structured as a collection of event handlers, which are procedure-like segments of code that are executed when a specified event occurs. Events are used to signify state changes of interest. When such an event occurs, all event handlers bound to that event are executed. Events can be raised explicitly by micro-protocols or implicitly by the runtime system. Execution of handlers is atomic with respect to concurrency, i.e., each handler is executed to completion without interruption. The runtime system also supports shared data (e.g., messages) that can be accessed by all micro-protocols.

The two most important event-handling operations are:

- **bind**(*event, handler, order, static_args*) - specifies that *handler* is to be executed when *event* occurs. *order* is a numeric value specifying the relative *order* in which *handler* should be executed relative to other handlers bound to the same event. When the handler is executed, *static_args* are passed as arguments.
- **raise**(*event, dynamic_args, mode, delay, urgency*) - causes *event* to be raised after *delay* time units. If *delay* is 0, the *event* is raised immediately. The occurrence of an event causes handlers bound to the event to be executed with *dynamic_args* (and *static_args* passed in the **bind** operation) as arguments.

Execution can either block the invoker until all handlers have completed execution (*mode* = SYNC) or allow the caller to continue (*mode* = ASYNC). *urgency* is a numeric value specifying the relative urgency with respect to other handlers already queued for execution. An event raised with *delay > 0* is also called a *timer event*.

Other operations are available for such things as creating and deleting events, unbinding an event handler, halting event execution, and canceling a delayed event. The event mechanism and shared data structures provide a degree of indirection between micro-protocols that facilitates the configuration of different collections into functional services. The model also has a number of

appealing features for real-time systems. One is that event handlers are short and simple, with predictable timing behavior. Another is that atomic execution of handlers minimizes the need for synchronization between handlers and, thus, reduces problems such as priority inversion.

The composite protocol

We have considered that the networking subsystem is constructed as a protocol graph. This graph can be divided between user and kernel space, with protocols in the latter having better predictability (e.g., no page faults) and faster performance (e.g., fewer context switches). In our design, one or more of the protocols in that graph are composite protocols. An application may open multiple logical connections called sessions through the protocol graph to other application level

objects. Each session has its own QoS guarantees, which provides a second, dynamic level of QoS customization. Scheduling of handlers within a composite protocol is implemented using an ordered event handler queue (EHQ) storing pointers to handlers awaiting execution. Handlers are added to the EHQ when an event to which they are bound occurs and removed when executed. The order of handlers in the EHQ is determined by the relative urgency of the corresponding event. Handlers are executed by a dispatcher thread that executes handlers serially. This explicit scheduling is necessary since all threads associated with a single path have the same priority and would otherwise be executed in some arbitrary order by the scheduler. Fig. 1 illustrates this design.

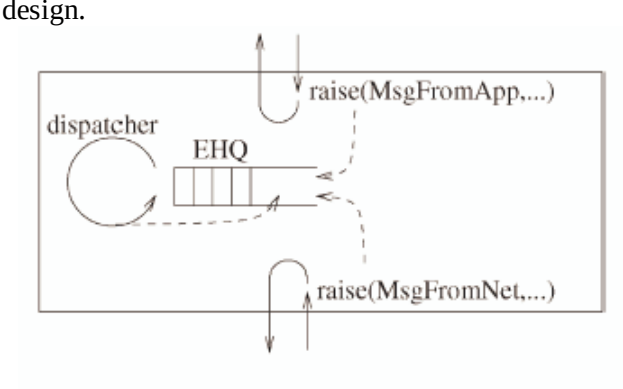

Fig.1. Composite protocol

In the fig. 1, solid arrows represent threads of execution, including the dispatcher thread associated with the EHQ. Protocols above and below the composite protocol interact with it through a procedure call *interface*–the x-kernel push and pop operations–and thus, the threads from these components operate within the composite protocol as well. The dashed arrows in the figure represent handlers being inserted into the EHQ as a result of event occurrences. In addition to the EHQ, each composite protocol contains an event/handler map (EHM) that maintains information about which handlers are bound to each event. Handlers for a given event are stored in this structure according to their relative ordering requirements.

As an example, an event handling pseudocode is shown in Fig. 2. The implementation of raise depends on the mode supplied as an argument. A nonblocking raise (ASYNC) simply inserts the handlers bound for this event into the EHQ, while a blocking raise (SYNC) causes the thread executing the raise to assume the role of the dispatcher until all the handlers for this event occurrence have completed. An event raised with *delay > 0* is implemented using the timer mechanism to realize the appropriate delay.

```
raise (event,dargs,mode,delay,urgency) {
   if(delay > 0)
      evSchedule(later,delay,event,dargs,urgency);
   else { handlers = EHM (event);
      addArgs(handlers,dargs);/*add dargs to handler
info*/
      if (mode = = ASYNC) insert(handlers);
      else (mode = = SYNC)
   }
}
void dispatcher() {
   while (true) {
     yield CPU to any higher priority thread;
     P(Sn); h:HandlerInfo = head(EHQ);
     call h→func(h→args); }
}
void insert(handlers,urgency) {
  insert handlers into EHQ ordered by urgency;
  Sn = number of handlers
}
```

Fig. 2. Event handling pseudocode

4 Possible Configurations and Resource Management Framework for the Wireless Network

An RTD channel can be customized across a wide range of QoS attributes, which allows it to satisfy applications with diverse real-time and dependability requirements. The customization of a channel typically starts by choosing the channel shape from the eight available. Next, the behavior of the channel is specified further by selecting the desired ordering, jitter, reliability, and timeliness. In addition to the discussed micro-protocols, RTD channels can be easily extended by designing and implementing new micro-protocols. This wide range of customizability allows the RTD channel service to be used in applications with diverse realtime and dependability requirements, including multimedia applications. In unidirectional video transmission, jitter control of the UD/PP or UD/MT channel is essential for transmission quality, but the message deadlines can be large. For distributed collaborative work applications such as PIP ordering and reliability

properties of a wireless channel are more important than tight jitter control or even very short deadlines since the consistency of every user's is essential. Currently, network management for a PIP system follow a platform-centric manager-agent paradigm compatible with the a centralised architecture with a single server node and radial clients.

In our scheme, the expected bandwidth is guaranteed to a connection while it remains in its initial cell. The difference between the desired and expected amounts of a connection is the actual borrowable bandwidth (ABB) and the cell may borrow some of this bandwidth from an existing connection in order to accommodate other incoming connections. The connection parameters are illustrated in Fig. 3. A new connection is accepted into a cell only if its expected bandwidth is less than or equal to the total bandwidth of the cell (not including the reserved handoff pool) divided by the number of connections in the cell. This requirement is actually a key feature of our scheme because it can force a connection to be blocked even if there is enough bandwidth for it in the cell.

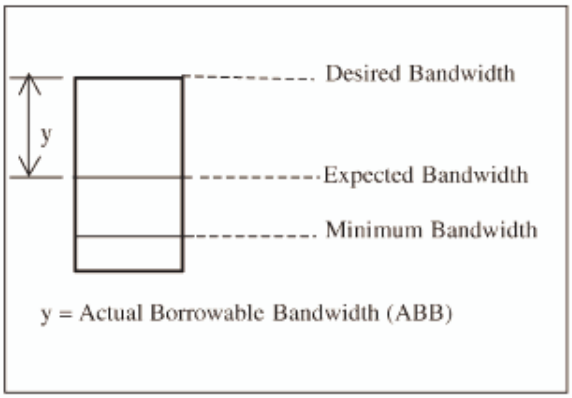

Fig. 3. Bandwidth requirements of a new connection.

If all the connections in a cell are functioning at their desired level and a new connection can also be accommodated at its desired level, it is simply admitted. If a connection cannot be given its expected amount using all the borrowable and free bandwidth in the cell, it is rejected. However, if it can be accommodated at its expected level, then the sum of the borrowable amounts of each connection plus any free bandwidth in the cell is divided equally among all the connections. Specifically, the equal share in a cell is determined by the formula below:

equal share = (total bandwidth) / (number_of_connections)

If a new mobile user requests a connection, the cell will accept or deny the connection based on the following algorithm:

```
if (expected bandwidth of new user ≤ equal share)
   if (expected bandwidth of new user ≤ total
       borrowable bandwidth of all users)
       accept connection
   else
       block call
```

When a connection terminates, the freed bandwidth is used first to replenish the reserved pool, with the leftover allocated to connections that are functioning below their expected level. Finally, any residual bandwidth is distributed in a max-min fair manner among the connections that are functioning below their desired level.

5 Preliminary Results, Expected Results and Future Work

Only a simple PIP application, that consist in still image segmentation, clustering, data block distribution, image processing on individual hosts and then processed blocks transfer and grouping in order to obtain a whole processed image was performed. The transfer of data was simulated on RTD channels. An RTD channel service including all the micro-protocols described above was already implemented, together with a simple admission control module. A number of experiments have been performed to test both functional and performance aspects of the service. All tests were run on an experimental platform with a cluster of 6 Pentium2 PCs connected via a dedicated 10 Mbps Ethernet hub, but the intention is to repeat the experiments in a wireless network.

For this purpose, a Simulation Model should be implemented. In order to evaluate our proposed schemes, we intend to simulate to kind of multimedia applications, with the traffic characteristics shown in the table 1.

Table 1 Traffic characteristics

Traffic content	Audio streams	Video streams
AVG BPS (kbps)	256	3000
MIN BPS (kbps)	256	1000
MAX BPS (kbps)	256	6000
AVG TIME (s)	300	600
MIN TIME (s)	60	300
MAX TIME (s)	1800	18000

The desired bandwidth and call length is generated for each new connection using a geometric distribution around the minimum, maximum, and average values given. The minimum bandwidth for each connection will be taken directly from the table, while the expected bandwidth will be set at half way between the generated desired amount and the given minimum amount. We also intend to give to each mobile host a speed characteristic (time spent in a cell) in order to simulate handoffs; thus, longer calls will likely experience more handoffs than shorter ones. We intend to use a 6 x 6 network of 36 cells, with a total bandwidth allocation of 30 Mbps for each cell. The mobile hosts (if exists) are considered moving in a directed manner. Specifically, a host is more likely to continue moving in the same direction than to turn. If a host reaches an edge of the network, the corresponding connection is terminated normally—hosts do not "bounce" back into the network.

The channels used in the preliminary experiment were simple BD/PP channels with no extra ordering, reliability, or jitter control micro-protocols. Table 2 shows performance results from experiments using one to four channels of varying priorities.

Table 2 . Roundtrip time distribution (in ms).

Channels	Priority	Average	90%	95%	100%
1	1	4.18	4.20	4.21	5.73
2	1	5.69	6.15	6.20	10.42
	2	7.80	8.23	8.28	12.90
3	1	6.25	6.72	7.41	10.26
	2	8.32	8.97	9.49	10.33
	3	10.56	11.21	12.14	27.85
4	1	7.09	7.69	7.88	23.82
	2	9.32	10.08	10.12	26.87
	3	11.40	12.70	12.84	34.94
	4	13.53	14.55	14.77	56.33

The times presented in the table 2 are roundtrip times in milliseconds and were calculated from measurements accumulated during 1,600 message roundtrips. The columns give the average over the 1,600 roundtrips, the 90th percentile value, the 95th percentile value, and the maximum. The numbers in the two percentile columns represent minimum possible deadlines that would be feasible for that value of P_D. Thus, for instance, for P_D = 95%, it would be feasible to choose a deadline of 5 ms for a single channel since a time of 4.21 ms has been measured. The numbers also show that the performance is predictable and generally consistent, especially for a user space implementation. Finally, the admission control module worked as expected in assigning priorities to channels and performing resource allocation decisions. In the final stage the admission control module will be tested with a variety of combinations of micro-protocols and channel shapes for functional correctness.

The next step is to move the RTD channel service into the kernel, which we expect will have a number of advantages. First, the performance of channels should improve considerably. Second, and more importantly, predictability should improve greatly. This can be used to prevent other system activity from interfering with the execution of RTD channels. Further, there is the possibility of abandoning the idea of a fixed reservation of bandwidth in favor of an adaptive or predictive reservation strategy.

References:

[1] B.M. Epstein and M. Schwartz, "Predictive QoS-Based Admission Control for Multiclass Traffic in Cellular Wireless Networks", *IEEE J. Sel. Areas in Comm.*, vol. 18, pp. 523-534, 2000.

[2] C. Oliviera, J. Kim, and T. Suda, "An Adaptive Bandwidth Reservation Scheme for High Speed Multimedia Wireless Networks", *IEEE J. Sel. Areas in Comm.*, vol. 16, pp. 858-874, 1998.

[3] K. Wang and L.-S. Lee, "Design and Analysis of QoS Supported Frequent Handover Schemes in Cellular ATM Networks", *IEEE Trans. Vehicular Techn.*, vol. 50, pp. 942-953, 2001.

[4] R. Bolla, F. Davoli, and M. Marchese, "Bandwidth Allocation and Admission Control in ATM Networks with Service Separation",*IEEE Comm.*, pp. 130-137, 1997.

[5] S.V. Raghavan and S.K. Tripathy, *Networked Multimedia Systems*, Prentice-Hall, 1998.

[6] R.Dobrescu, M. Dobrescu, "Reactive Congestion Control for Multipoint Video Services", 8th *International Workshop on Systems, Signals and Image Processing*, Bucureşti, Mai 2001

[7] K. Birman, A. Schiper, and P. Stephenson, "Lightweight Causal and Atomic Group Multicast", *ACM Trans. Computer Systems*, vol. 9, no. 3, pp. 272-314, Aug. 1991.