Multimedia Service Configuration and Reservation in Heterogeneous Environments *

Dongyan Xu, Duangdao Wichadakul, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
{*d-xu, wichadak, klara*}*@cs.uiuc.edu*

Abstract

Widely deployed multimedia services are expected to accommodate clients in a highly heterogeneous environment. Clients of a multimedia service may vary greatly in processing and communication capabilities. In addition, due to workload, location, and service time differences, the availability of end-to-end resources between a client and a server may also vary. Current solutions tend to focus on either the qualitative heterogeneity (in client and resource types) or the quantitative heterogeneity (in resource availability) problem. In this paper, we present a framework for dynamic end-to-end multimedia service configuration and reservation - an integrated solution to both aspects of the heterogeneity problem. Service configuration is responsible for choosing appropriate service components to compose a customized service delivery to each client; while service reservation is responsible for reserving the end-to-end resources in a coordinated manner, and providing the best possible quality within the chosen configuration. We have implemented a prototype of this framework as part of the 2K operating system, and tested it by building a proof-of-concept video streaming service on top of it. Our experiments show the soundness of this framework.

1 Introduction

The deployment of multimedia services is becoming ubiquitous. Clients of various capacity in a wide-area and heterogeneous environment will all be able to access multimedia services such as audio/video-on-demand, digital library, remote camera surveillance, and distributed visual tracking. In this paper, we propose a systematic approach to the provision of multimedia services in heterogeneous environments. More specifically, we address the challenges rising from the heterogeneity in multimedia client and server capabilities, and their end-to-end resource availabilities. For example, clients of a multimedia service may range from supercomputers to commodity PCs and smart handheld devices such as palm-tops. The network connections between the server and clients may range from high speed LANs to low speed dial-ups, from wireline to wireless. Furthermore (and less addressed), even for clients with the same machine type and connection type, the amounts of resources available to each of them may still vary, depending on their location, workload, and the time they make service requests. In particular, the bottleneck resource in each client's resource requirement may be different. Therefore, to deal with the heterogeneity problem, any solution that only targets one specific type of bottleneck resource (for example, the network) may not be effective in all situations.

We identify two co-related key issues in multimedia service provision in heterogeneous environments: service configuration and service reservation.

- Service configuration: how to choose appropriate service components along the path from the multimedia server to the client, and *possibly* via an intermediate gateway, so that the service can be delivered by the selected components with satisfactory end-to-end quality. Service configuration primarily deals with *qualitative heterogeneity* in client and resource types.
- Service reservation: for a chosen service configuration, how to properly reserve resources needed by the service components to achieve the best possible end-to-end QoS with resource efficiency. Service reservation should be performed in a coordinated manner due to its distributed and multi-resource nature, and it deals with *quantitative heterogeneity* in resource availability.

*This work was supported by the National Science Foundation under contract number 9870736, the Air Force Grant under contract number F30602-97-2-0121, National Science Foundation Career Grant under contract number NSF CCR 96-23867, NSF PACI grant under contract number NSF PACI 1 1 13006, NSF CISE Infrastructure grant under contract number NSF EIA 99-72884, NSF CISE Infrastructure grant under contract number NSF CDA 96-24396, and NASA grant under contract number NASA NAG 2-1250.

Most current solutions deal with only one aspect of heterogeneity. To integrate these solutions, we propose an integrated multimedia service management framework for dynamic end-to-end service configuration and reservation. The key entity in our framework is the *QoSProxy* running on each host. Every multimedia service session is set up by the relevant QoSProxies in a two-level iterative procedure, using the *service configuration protocol* and the *service reservation algorithm*.

We have implemented a prototype of this framework as part of the 2K system - a component-based distributed operating system for the new Millennium [10]. We will show the soundness of this framework with a CORBA-compliant proof-of-concept video streaming service, which is able to accommodate heterogeneous video clients (ranging from workstations to lap-top and palm-top computers) under various resource availability conditions. Furthermore, our experiments show a higher success rate for service reservations using a *contention-aware* resource reservation policy.

The rest of the paper is organized as follows. In Section 2, we describe the overall architecture of the proposed framework. In Section 3 we present the service configuration protocol and the service reservation algorithm executed by the QoSProxy. In Section 4, we describe our implementations of the framework prototype and the example video streaming service, and evaluate their performance. Section 5 discusses related work. Section 6 concludes this paper.

2 Overall Architecture

The overall architecture of the integrated multimedia service management framework is shown in Figure 1. The framework encompasses *service components*, *QoSProxies*, and *resource brokers*. Service components are the basic building blocks to compose multimedia services. A QoSProxy is deployed on each host. Multiple resource brokers are running on each host, one for each type of resource. The QoSProxy interacts with the local service components and resource brokers, as well as QoSProxies on other hosts. The roles of these entities are described in the following subsections.

2.1 Service Components and Configurations

Our framework encourages developers to implement a multimedia service as reusable service components and their combinations, rather than a single monolithic program. A service component is a functional unit participating in a service, and multiple components work together to deliver a service. Furthermore, it is often possible that the same type of service can be delivered by *different* combinations of components. This is especially necessary in a heterogeneous environment where client capability and end-to-end

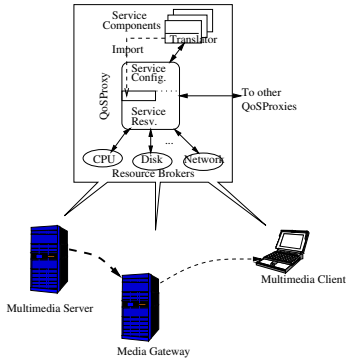

Figure 1. Overall Architecture

resources availability vary. Therefore, service configuration is to choose the suited service components on the server, on the client, and possibly on an intermediate gateway, in order to deliver the service to the client with satisfactory quality.

Figure 2 shows an example of multiple service configurations for the same type of service. The available service components on the server side include c_a, c_b and c_c; the service components on the clients include c_d, c_e, or c_f. In addition, there is a service component c_g on a media gateway that may also participate in this service. The Figure shows four possible end-to-end service configurations C_1, C_2, C_3, and C_4. Arrows between the components indicate the directions of the media streams. For client I, C_1, C_2, and C_3 can be the candidate service configurations; for client II, there are C_1 and C_2; for client III, the only possibility is C_4.

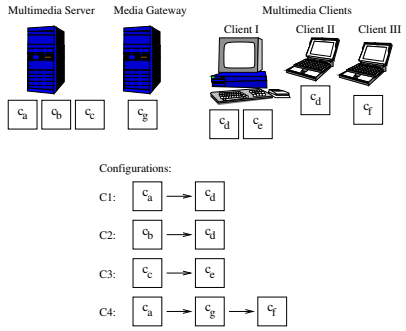

Figure 2. Example of Service Components and Configurations

The framework also requires (and defines uniform interface for) a *translator* to accompany each service component. The translator is responsible for translating multimedia service quality into system level resource requirement. The resource requirement specifies the types and quantities of system resources needed by a service component. As will be described in Section 2.2, translators are dynamically imported by the QoSProxy, and serve as the bridge between the service configuration and reservation tasks. Note that a

translator is highly dependent on the service component's implementation. For example, it is possible that the MPEG encoder components by company X and company Y have different resource requirements for the *same* level of service quality.

2.2 QoSProxy

The QoSProxy is the key entity in the framework. It can be logically organized in two layers: the service configuration layer for end-to-end service configuration, and the service reservation layer for end-to-end service reservation.

2.2.1 End-to-End Service Configuration

The QoSProxy executes the *service configuration protocol* to select the proper set of service components along the end-to-end path for each service request. Each QoSProxy maintains a *Service Component Table* with information about the registered *local* service components. Furthermore, the QoSProxy of each multimedia server maintains 'knowledge' about the candidate service configurations and the service component dependencies in each configuration. The knowledge is stored in *Configuration Tables*, one table for each type of multimedia service.

In a Configuration Table, each entry contains a dependency graph for one candidate service configuration. Furthermore, the Configuration Table is organized such that the smaller the index of a candidate service configuration, the higher the priority of its selection. Priority rankings of all the candidate configurations are also part of the knowledge specified by the service provider. The reasons for this design include: (1) the service provider is more likely to have the expertise and incentive to accommodate heterogeneous clients, and a server is typically dedicated to one or just a few types of services; and (2) clients may be simple and wish to be free from fairly complicated service management, and they may request any type of service. As a result, the client-side QoSProxy is 'thinner' at the service configuration layer than the server-side QoSProxy.

2.2.2 End-to-End Service Reservation

Once a service configuration has been chosen, the server-side QoSproxy initiates the end-to-end service reservation, which is performed cooperatively by each QoSProxy whose local service component(s) is in the chosen configuration. In order to compute the resource requirement of each service component, the QoSProxy imports the corresponding translator of the component (as shown in Figure 1). The QoSProxy then dispatches the translated resource requirement to each *resource broker*, which performs the individual resource reservation.

2.3 Resource Broker

A resource broker is responsible for the monitoring, reservation, and scheduling of a certain type of system resource. Different resource brokers have been developed independently for resources such as CPU, network, disk, and memory. For example, the QualMan system [8] includes QoS-aware resource brokers for CPU, network, and memory, respectively; and Cello [9] provides a disk scheduling framework. Before any resource broker can be integrated into our framework, it is necessary to put a thin 'wrapper' outside the broker's native interface. This allows the QoSProxy to interact with different resource brokers using a uniform interface. The interface includes at least the following operations: (1) reporting current resource utilization and availability, (2) reserving resource, (3) releasing resource, and (4) reporting possible reservation degradation.

3 Operations in the Framework

After introducing the overall architecture, we now describe the operations within the framework, which include the service configuration protocol and the service reservation algorithm.

3.1 Service Configuration Protocol

The service configuration protocol is executed by QoSProxies at the service configuration layer as shown in Figure 3. The protocol steps are as follows:

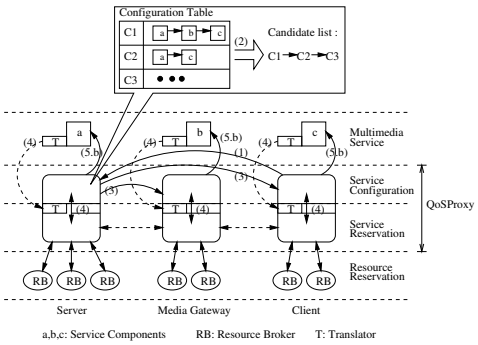

Figure 3. Service Configuration Protocol (current configuration is C_1)

Step (1): When a client makes a multimedia service request, the client-side QoSProxy forwards the service request to the server-side QoSProxy, together with the names of client-side service components that may participate in this service (for example, the set of available media players).

Step (2): The server-side QoSProxy looks up its Configuration Table for the requested service, pulls out a *list of*

candidate service configurations, whose client-side component matches one of the component names passed by the client-side QoSProxy. The server-side QoSProxy chooses head of the list as the current candidate configuration.

Step (3): The server-side QoSProxy notifies the client-side QoSProxy of the current candidate configuration. If a media gateway is involved in the candidate configuration, the QoSProxy of the media gateway is also notified[1].

Step (4): Each QoSProxy involved imports the translator(s) of its local service component(s) in the current configuration, and executes the service reservation algorithm. Depending on the feedback from the service reservation layer, the protocol proceeds to either Step (5.a) or (5.b).

Step (5.a): If the service reservation algorithm fails, The server-side QoSProxy chooses the next candidate configuration from the list. If there is no more candidate, it reports service request failure to the client-side QoSProxy; otherwise the protocol returns to step (3).

Step (5.b): If the service reservation algorithm succeeds, each QoSProxy will start its local service component(s) at the contracted QoS level returned by the service reservation algorithm.

3.2 Service Reservation Algorithm

The service reservation algorithm is executed by the QoSProxies at the service reservation layer. The main objectives of this algorithm are to deal with heterogeneity in end-to-end resources availability, and to achieve the best end-to-end QoS within a chosen service configuration. More specifically, the following issues need to be considered: (1) the relation between the service quality and its resource requirements is non-linear, both of which are generally expressed as *partially ordered multi-dimensional vectors*; (2) the success of a service reservation depends on the success of *every* required resource's reservation; and (3) any resource could become the bottleneck in multi-resource reservations. Our solution to these issues is based on a general *QoS-aware resource model*, and a *contention-aware reservation policy*.

3.2.1 A QoS-Aware Resource Model

In the QoS-aware resource model, each service component c is associated with a pair of Q^{in} and Q^{out} - the quality of its input and output data, respectively. Q^{in} and Q^{out} are vectors of multiple quality parameters (however, Q^{in} and Q^{out} may not have the same parameter list). QoS vectors with the same parameter list can only be ranked in a *partial order*. As is often the case in practice, we assume that each parameter takes *discrete* values, therefore the QoS vectors for Q^{in} and Q^{out} are enumerable.

The translator of a service component is defined as a function T ($T : Q^{in} \times Q^{out} \rightarrow R$) which computes the required resources R to achieve Q^{out} with the presence of Q^{in}.

$$R = T(Q^{in}, Q^{out}) \quad (1)$$

where resource vector $R = [r_1, r_2, ...r_M]$, and $r_m (1 \leq m \leq M)$ is the required quantity of the mth resource. Similarly, the resource vectors computed by the same translator can be ranked in a partial order.

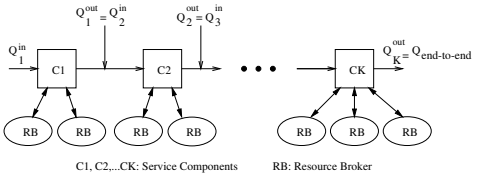

Figure 4. Quality Dependencies of Service Components in a Configuration

In this paper, we assume that the service components in a service configuration have linear quality dependencies as shown in Figure 4 (extension to the linear dependencies assumption is our on-going work). The Q^{in} of $c_k, (1 < k \leq K)$ is the Q^{out} of c_{k-1}. Q^{in} of c_1 is the original quality of the source multimedia data (for example, a stored or live video source). The Q^{out} of c_K is the end-to-end QoS achieved by this service configuration. For presentation simplicity (without decreasing the problem's complexity), we further assume that (1) the Q^{in} of c_1 takes a single vector, i.e. the quality of the source data is fixed; and (2) the Q^{out} vectors of c_K can be ranked into a total order, which reflects the client's preferences (for example, when two Q^{out} vectors are not comparable, the one with shorter end-to-end delay is better).

3.2.2 A Contention-Aware Reservation Policy

Each resource needed in a service configuration may also be requested by others, and the degree of resource contention varies from time to time, resource to resource. We introduce a dynamic, contention-aware resource reservation policy as follows.

Let r be the required amount of a resource computed by a translator, and r^{avail} be the currently available amount of this resource reported by the resource broker. We first define a metrics to evaluate how 'competitive' it is to reserve r under the availability of r^{avail}. We choose a simple metrics λ as follows (there are other possible metrics):

$$\lambda = e^{N*u} \times \frac{r}{r^{avail}}, \quad (r \leq r^{avail}) \quad (2)$$

[1] The location of the media gateway can be dynamically discovered by *MeGaDiP*, a wide-area and resource-aware media gateway discovery protocol [11].

where N and u are also provided by the resource broker: u is the recent average utilization of this resource and $N(\geq 0)$ is a resource-specific constant[2]. We call λ the *contention index*. Intuitively, the larger the portion ($\frac{r}{r^{avail}}$), the less likely the reservation will succeed with the presence of other requesters. Furthermore, the higher the recent utilization (u), the larger the current resource demand can be - we assume an *exponential* increase in resource demand with the growth of u. Therefore, λ is a reasonable indication of the potential contention faced by the reservation request for r. Correspondingly, for the resource vector $R = [r_1, r_2, ...r_M]$ associated with a service component c, let

$$\Lambda = MAX_{m=1}^{M} \lambda_m = MAX_{m=1}^{M} (e^{N_m * u_m} \times \frac{r_m}{r_m^{avail}}) \quad (3)$$

be the contention index of c. Hence, for a service configuration with service components $c_1, c_2, ...c_K$, if $\Lambda_k, (k = 1, 2, ...K)$ is the contention index of service component c_k, then in order to minimize the end-to-end reservation contention with other requesters, our contention-aware policy is to *minimize* $MAX_{k=1}^{K} \Lambda_k$, under the constraint of the best achievable end-to-end QoS by the service configuration.

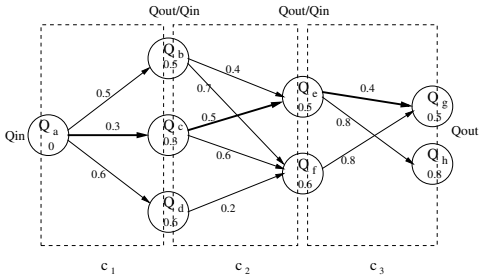

Figure 5. An Example of the Service Reservation Algorithm (Λ values are shown)

We use an example to illustrate our service reservation algorithm which is based on this policy. Figure 5 shows a *QoS-resource graph* for a three-component service configuration. *Notice the significant representation differences from previous Figures*: here, each dotted rectangle represents a service component; each node represents a QoS vector; and each directed edge from node Q_X to Q_Y represents resource vector $R = T(Q_X, Q_Y)$. The edge for R exists if and only if $R \leq R^{avail}$, and vector R^{avail} represents the available amounts of resources required by that component. The corresponding Λ value is the edge's weight, as shown in the Figure.

We can see that both Q_g and Q_h are achievable end-to-end QoS vectors, because there exist paths from node Q_a to Q_g and to Q_h, and each path represents a possible reservation plan under the current resources availability. Suppose that the client prefers Q_g to Q_h, then Q_g is the best achievable QoS, and our reservation policy is reduced to the following problem: "*find the shortest path from Q_a to Q_g* - if we *re-define* the '+' operation as the 'MAX' operation". Figure 5 shows such a (thicker) path. Then, *backtracking* this path, resources will be reserved for each component (from c_3 back to c_1) according to the resource vector represented by each edge on the path.

The service reservation algorithm is executed by the QoSProxies in a distributed fashion. The computational complexity of this algorithm is $O(K\mathcal{Q}^2)$, where K is the number of service components in a service configuration, and $\mathcal{Q}$ is the maximum number of candidate QoS vectors as the input or output quality for any service component. For example, in Figure 5, $\mathcal{Q} = 3$, which is the number of candidate QoS vectors as the output (input) quality for c_1 (c_2). Fortunately, K and $\mathcal{Q}$ usually have fairly small values in practice.

4 Prototype Implementation and Performance

We have implemented a prototype of the integrated framework as part of the 2K system - a component-based distributed operating system for flexible configuration and adaptive execution of distributed objects. Our prototype has been implemented as CORBA objects.

4.1 Framework Prototype Implementation

Our prototype implementation is outlined in Figure 6. We use the CORBA *Naming*, *Trader*, and *Property Services* to organize and manage the entities in our framework, including the service components, the translators, the QoSProxies, and the resource brokers.

To demonstrate the soundness of the framework, we have also implemented a video streaming (VS) service on top of it. The VS clients in our testbed include workstations and palm-top devices (3COM PalmPilot). They vary greatly in CPU and memory capacities, and they are connected to the VS server via network connections of different bandwidth. Furthermore, their client-side VS components are different. The workstations are installed with one or more of the following: a MPEG-I player, a simulated low bit-rate player (called SimH261 player), or a simple bitmap player; the palm-top we use only have the simple bitmap player. However, the VS server only has stored videos in MPEG-I format.

A QoSProxy is deployed on each host in the testbed. In order to deal with the heterogeneity, we provide configuration knowledge to the QoSProxy of the VS server. Specifi-

[2]The resource broker uses N to indicate the degree of sharing for this resource. For example, a server's CPU may have a higher N than a client's CPU.

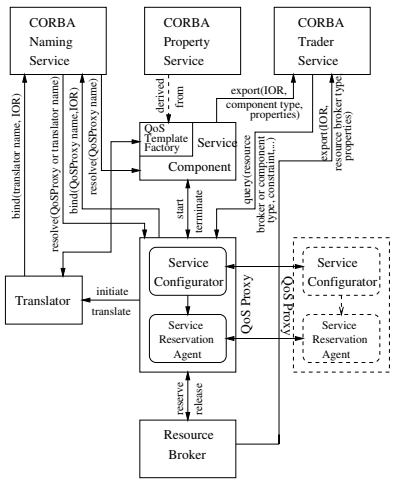

Figure 6. Framework Prototype Implemented as CORBA Objects

cally, we fill its Configuration Table with the following candidate configurations:

- C_1 (end-to-end streaming): this configuration involves two service components: *MpegSender* on the server and *MpegPlayer* on the client. We apply video streaming methods such as smoothing [4], buffering, and selective frame dropping to achieve different levels of QoS under various resources availability conditions.

- C_2 (streaming via MPEG-H.261 transcoder): this configuration involves three service components: *MpegSender* on the server, *SimH261Player* on the client, and *SimMpegH261Transcoder*, a simulated MPEG-to-H.261 transcoder on a media gateway. This configuration targets clients with a low bandwidth or congested connection. Video streaming methods similar to the ones in C_1 are also applied.

- C_3 (streaming via MPEG-Bitmap transcoder): this configuration involves three service components: *MpegSender* on the server, *BitmapPlayer* on the client, and *MpegBmpTranscoder* [6] on the media gateway. This configuration targets clients with very limited CPU and memory (for example, the palm-top in our testbed). There is very little buffering on the client due to its limited memory.

Note that the streaming techniques in these service configurations are not new. Our purpose here is to demonstrate how these configurations can be organized and dynamically selected in a systematic manner using our framework.

4.2 Performance Evaluation

We have performed a number of experiments and simulations based on the prototype. The VS server in our testbed is a Sun Ultra-2 workstation with two UltraSPARC 200MHz processors and 256MB memory, running Solaris 2.6 operating system. The media gateway is a 266MHz Pentium II PC with 128MB memory, running Windows NT 4.0. Each client is either (1) a Sun Ultra-1 workstation with a UltraSPARC 143Mhz processor and 128MB memory running Solaris 2.6; or (2) a Palm III running PalmOS 3.0 with 8MB memory. The server and the media gateway are on the same 10Mbps Ethernet. The palm-top is connected to the media gateway via a serial line of 56.6kbps. One workstation client is on the same LAN as the server, and the other workstation client is two hops away from the server[3]. The video file used in the experiments is a MPEG-1 clip with a resolution of 352x240 pixels and a recording rate of 24fps.

4.2.1 End-to-End Service Quality

We will show that our framework is able to provide very flexible levels of end-to-end QoS to heterogeneous clients, under various resources availability conditions. We perform the following sets of experiments, each set assuming a different bottleneck resource. The value of N in the contention index for each resource is 1.0, and we set different r^{avail} values of the bottleneck resource[4]. We load all three player components on the workstation client two hops away from the server. Then the client requests VS service under different r^{avail} values of the bottleneck resource.

First, we assume that the bottleneck resource is the bandwidth of the LAN to which the client is connected. Table 1 shows (1) the amount of available network bandwidth, (2) the service configuration to be chosen and the QoS vector (frame rate, end-to-end delay) computed by the service configuration protocol, and (3) our measurement of the actual service quality during the service execution.

Second, we assume that the bottleneck resource is the CPU capacity of the client. Table 2 shows the results. For the same frame rate, the bitmap player requires less CPU than the simulated H.261 player, which requires less CPU than the MPEG player. Therefore, the configuration changes from C_1 to C_2 to C_3, with the decrease of client CPU capacity.

The results demonstrate wide ranges of achievable end-to-end QoS - both within a service configuration and in different service configurations. Notice that the actual service quality is more or less different from the quality computed by the service configuration protocol. The reason is that

[3]We are implementing a RSVP-based network broker. Currently, bandwidth reservation is simulated by end-to-end measurement and flow control in our networks with light external load.

[4]For simplicity, we ensure that other resources are always sufficient.

the translators we implement only use a very rough profile (frame rate, average and maximum frame sizes) derived from the source MPEG file, when computing the amounts of required resources. Therefore, the amounts of required resources are not accurate, affecting the actual service quality.

Available Bandwidth	Cfg.	Computed QoS	Measured QoS
2.5Mbps	C_1	24.0fps, 200ms	21.2fps, 178ms
1.8Mbps	C_1	24.0fps, 7.0s	22.0fps, 9.8s
1.8Mbps	C_1	16.0fps, 200ms	12.8fps, 165ms
1.0Mbps	C_1	16.0fps, 7.0s	14.4fps, 10.3s
1.0Mbps	C_1	8.0fps, 200ms	6.6fps, 159ms
0.3Mbps	C_1	8.0fps, 7.0s	8.0fps, 10.1s
64Kbps	C_2	24.0fps, 280ms	21.3fps, 228ms
32Kbps	C_2	24.0fps, 12.0s	21.5fps, 17.4s
32Kbps	C_2	16.0fps, 280ms	14.0fps, 213ms
32Kbps	C_3	8.0fps, 250ms	6.4fps, 197ms
16Kbps	C_3	4.0fps, 250ms	4.0fps, 193ms

Table 1. Multiple End-to-End QoS Levels with Different Bandwidth

Available CPU	Cfg.	Computed QoS	Measured QoS
80%	C_1	24.0fps, 200ms	24.0fps, 172ms
60%	C_1	16.0fps, 200ms	14.8fps, 152ms
40%	C_1	8.0fps, 200ms	8.0fps, 180ms
40%	C_2	24.0fps, 280ms	24.0fps, 218ms
25%	C_2	16.0fps, 280ms	15.6fps, 252ms
15%	C_2	8.0fps, 280ms	8.0fps, 193ms
10%	C_3	8.0fps, 250ms	8.0fps, 200ms
5%	C_3	4.0fps, 250ms	4.0fps, 198ms

Table 2. Multiple End-to-End QoS Levels with Different Client CPU

4.2.2 Service Reservation Success Rate

In this section, we evaluate the success rate of end-to-end service reservations achieved by the service reservation algorithm. Unlike the previous section which evaluates performance from the point of view of an individual client, we evaluate the success rate of service reservations in the presence of multiple clients. Due to the total resources limitation in our testbed, the results in this section are obtained from simulation.

We simulate the simple scenario which involves one server, one gateway, and multiple clients. We assume that all clients' service requests are configured using the same configuration, which includes service components c_1 on the server, c_2 on the gateway, and c_3 on each client. The QoS-resource graph for this configuration is shown in Figure 7. For simplicity, we assume that each service component only requires one single resource. Each value in brackets by the edge is the required amount of resource for the corresponding (Q^{in}, Q^{out}) pair. The total amount of resource on the server is 800 units, with an N value of 2 and an initial utilization of 75%; the total amount of resource on the gateway is 300 units, with an N value of 1 and an initial utilization of 50%; and the total amount of resource on each client is 1 unit, with an N value of 0.1 and an initial utilization of 0%. Each client makes one service request to the server. The request time is uniformly distributed within an one minute period starting from time 0. We also assume that for each client, right before it makes the service request, a background task will begin to run with a 0.5 probability, and the amount of resource it occupies is uniformly distributed between 0.25 and 0.75 unit. We assume the worst case that during the one-minute period, no resource is released on the server, gateway, or client.

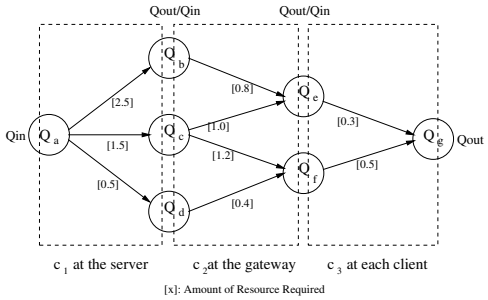

Figure 7. QoS-resource Graph of the Simulated Configuration

Figure 8 shows the success rate of service reservations with increasing number of clients. We compare our service reservation algorithm based on the contention-aware reservation policy, and a 'random' algorithm which randomly selects a reservation plan among all the paths in Figure 7. The results show that by using the contention-aware policy, our service reservation algorithm constantly achieves higher success rate than the random algorithm.

5 Related Work

There have been research works on the issue of supporting multimedia services in heterogeneous environments. Most of the proposed solutions fall into *one* of the following categories: (1) performing quality adaptation within the same service configuration, by providing service components with adaptive techniques (such as layered media [7], or feedback-based control [2]); and (2) employing differen-

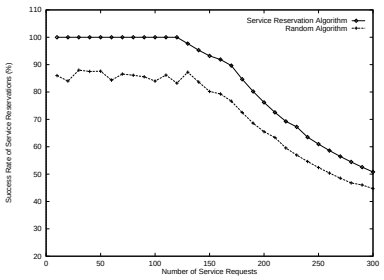

Figure 8. Service Reservation Success Rate

t service components (for example, media transcoders [1]) to adapt to network and client variability. Solutions in the first category primarily deal with quantitative heterogeneity, while solutions in the second category are more concerned about qualitative heterogeneity. In this paper, we introduce a general framework *integrating* the solutions in both categories, instead of proposing another specific solution. Furthermore, our framework does not assume a specific resource as the bottleneck resource.

Our framework takes one step up from resource management to end-to-end service management. Like a resource, a multimedia service can be configured and reserved as a whole. In the Darwin project [3], the concept of value-added service is suggested, and an application-oriented and hierarchical service brokerage architecture is introduced. However, it focuses more on composing sophisticated value-added service, rather than dealing with heterogeneity in multimedia service provision. In addition, unlike our service reservation algorithm, the Beagle signaling protocol in Darwin is not contention-aware. In [5], the concept of open binding is introduced for the implementation of resource and service management mechanisms for adaptive multimedia applications. However, as a meta-framework, open binding itself does not provide any multimedia service management protocol or algorithm, contrary to our framework.

In this paper, we have focused on a reservation-based environment, where each type of resource can be reserved. Such an environment has become more available, with the development of resource management and resource brokerage techniques (for example, the real-time extension to general OS, and the differentiated service on the Internet). On the other hand, in [12], we also study multimedia service configuration *without* assuming a reservation-based environment.

6 Conclusion

We have presented an integrated framework for end-to-end multimedia service configuration and reservation in a heterogeneous environment. QoSProxy, the key entity in the framework, interacts with both the application level multimedia service components and the underlying resource brokers, in order to provide heterogeneous clients with the best possible end-to-end QoS. We have described the executions of (1) the service configuration protocol, which selects the best suited service configuration for each individual client; and (2) the service reservation algorithm, which reserves end-to-end resources in a coordinated and contention-aware manner for a chosen service configuration. We have also presented experimental results obtained from our implementation of the framework's prototype, as well as a proof-of-concept video streaming service built on top of it. The results show that our framework is able to provide heterogeneous clients with a wide range of service quality choices under different resources availability conditions. Furthermore, service reservations enjoy high success rate.

References

[1] E. Amir, S. McCanne, and H. Zhang. An application level video gateway. *Proceedings of ACM Multimedia'95*, November 1995.

[2] S. Cen, C. Pu, and J. Walpole. Flow and congestion control for Internet streaming applications. *Proceedings of SPIE/ACM MMCN'98*, January 1998.

[3] P. Chandra, A. Fisher, C. Kosak, T. Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Resource management for value-added customizable network service. *Proceedings of IEEE ICNP'98*, October 1998.

[4] W. Feng and J. Rexford. Performance evaluation of smoothing algorithms for transmitting prerecorded VBR video. *IEEE Trans. on Multimedia*, September 1999.

[5] T. Fitzpatrick, G. Blair, G. Coulson, N. Davis, and P. Robin. Supporting adaptive multimedia applications through open bindings. *Proceedings of International Conference on Configurable Distributed Systems*, May 1998.

[6] C. Hess, D. Raila, R. Campbell, and D. Mickunas. Design and performance of MPEG video streaming to palmtop computers. *Proceedings of SPIE/ACM MMCN 2000*, January 2000.

[7] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. *Proceedings of ACM SIGCOMM'96*, August 1996.

[8] K. Nahrstedt, H. Chu, and S. Narayan. QoS-aware resource management for distributed multimedia applications. *Journal of High Speed Networks*, 7(3), 1998.

[9] P. Shenoy and H. Vin. A disk scheduling framework for next generation operating systems. *Proceedings of ACM SIGMETRICS'98*, June 1998.

[10] The 2K Team. The 2k project. *http://choices.cs.uiuc.edu/2K*.

[11] D. Xu, K. Nahrstedt, and D. Wichadakul. MeGaDiP: a wide-area media gateway discovery protocol. *Proceedings of IEEE IPCCC 2000*, February 2000.

[12] D. Xu, D. Wichadakul, and K. Nahrstedt. Resource-aware configuration of ubiquitous multimedia services. *In submission*.