A Gateway-Assisted Approach Toward QoS Adaptations

William Kalter, Baochun Li, Won Jeon, Klara Nahrstedt, Jun-Hyuk Seo*
Department of Computer Science
University of Illinois at Urbana-Champaign
kalter,b-li,wonjeon,klara,jseo@cs.uiuc.edu

Abstract

In this paper, we present a gateway-assisted QoS adaptation framework which satisfies high-level QoS application guarantees. We validate this framework via a distributed tracking system and show provisioning of critical QoS guarantees despite the best effort underlying resource management.

Keywords*: Visual tracking, resource adaptation, adaptive middleware, gateway*

1. Introduction

Many distributed client/server applications today execute over a best-effort system environment. In this environment the underlying system is incapable of providing resource guarantees, such as network bandwidth and processor time. Despite this lack of guarantees we would like the application to execute using the current resource availability. To do this, the application should adapt itself to execute at a level which provides QoS that is acceptable to the user yet utilizes only as much resources as are currently available.

Several recent works have attempted to perform these adaptations for distributed client/server applications. The first approach, taken in [1, 4, 7], is to create a QoS-aware proxy between the client and server. This proxy transcodes the server data so that it matches the client's reconfigured QoS needs, thus enabling the client to dynamically alter its execution level [2]. The second approach is to provide a gateway between the client and a series of servers [5]. In this approach, the gateway selects one server from a series of servers for each client, with each server providing an equivalent data stream. The first approach provides QoS correlated data between the client and server but limits reconfigurations to those which can be performed on the outgoing server data stream. Since the server is not involved in the reconfiguration process it is not possible to adapt the QoS of the stream content itself. In contrast, the second approach allows for the possibility of providing a different QoS satisfying application data stream from each server. However, it does not provide a means for reconfiguring the format of the data since each server must provide equivalent service to the client.

In this paper, we present a framework which merges these two approaches to provide reconfiguration capabilities for both the application data content and the resources required to transport and present data to the client. Our framework is centered around an intelligent gateway which facilitates distributed QoS reconfigurations for a client. This gateway exists as an extension of the adaptive middleware architecture previously presented in [6].

The remainder of this paper is organized as follows: Section 2 describes the design of the gateway framework. Section 3 describes a particular application of the framework, the *omni-directional video camera* (ODVC). Section 4 describes the implementation of our framework. Section 5 shows some experimental results from the ODVC application. Section 6 then concludes this paper.

2. Design

2.1. The Gateway-Centric Architecture

The gateway-centric architecture used by our framework consists of a group of servers and clients connected to a gateway. Figure 1 shows the toplogy of nodes within the framework. The gateway manages control connectivity between each client and the servers and assists the client in QoS adaptive behavior. Note that the transfer of data will follow a direct path between one client and one server.

The gateway and servers are collectively coordinated and serve as a *single service facility*. At any time, each client is being serviced by a single server within the facility. The decision as to which server is serving the client is dynamically chosen based on client resource observations, user per-

*This research was supported by the NSF PACI grant under contract number NSF PACI 1 1 13006, NSF Career under contract number NSF CCR 96-70736, NSF CISE Infrastructure grant under contract number NSF EIA 99-72884, ARPA grant under contract number F30602-97-2-121, NSF CISE Infrastructure grant under contract number NSF CDA 96-24396, and NASA grant under contract number NASA NAG 2-1250.

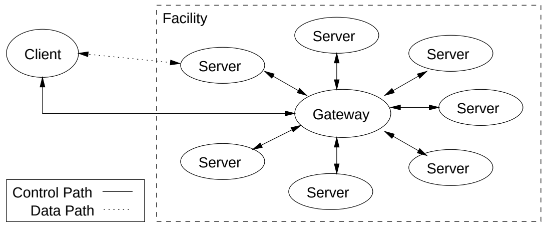


Figure 1. Topology of client and facility

ceived observations, and gateway state information.

The ODVC provides a typical example of a single service facility. Here, the video servers are arranged circularly around common scene of interest. Each client receives a video stream only from a single server within the facility. The decision which server streams video to the client is made by the client, depending on which view of the scene the user wants to see.

To assist QoS functional adaptations, we designed a set of protocols within the single service facility, as well as between the clients and the service facility. The goals of our design are:

1. *Agility*. The gateway should be able to maintain a dynamic environment in which servers are added, removed, and unexpectedly fail.
2. *Speed*. The amount of time taken to switch servers for a client should be kept minimal, growing no worse than linearly in the number of servers within the facility.
3. *Flexibility*. The framework should be generic so that a variety of new applications can be easily extended from it. Any application-specific QoS information should be efficiently handled by the framework and then interpreted using an application-specific knowledge base.
4. *Development transparency*. There should be very little difference between developing an application with one highly reconfigurable server and an application with several less reconfigurable servers which can be dynamically switched by the gateway.
5. *User transparency*. The end user should neither be aware of nor involved in the negotiation and switching protocols taking place within the middleware.

The next three sections focus on the server-gateway protocols, client-gateway protocols, and internal gateway protocols.

2.2. Server-Gateway Protocols

As previously stated, each server can be designed as though it were the only server in the standard single client/server model. To make the server perform within a facility should require only a minimal set of additions to the server. This is accomplished by adding two component entities to the server middleware: an observer and a negotiator.

The observer monitors local resource conditions which affect the current utility of the server, such as CPU load. The server then periodically updates the gateway with the state of these conditions.

The negotiator maintains communication between the server and the gateway. These communications include informing the gateway that the server has not died, keeping the gateway apprised of QoS conditions satisfied by the server, and responding to client authentication requests made by the gateway.

2.3. Client-Gateway Protocols

The major goal of the client in this protocol is to observe QoS and adapt with large scale adaptations in case of QoS degradation. The adaptation performed is decided by a two-tier adaptive middleware as shown in Figure 2 [6]. We will briefly discuss this architecture in order to validate its use as the driving force behind the QoS adaptations at the gateway.

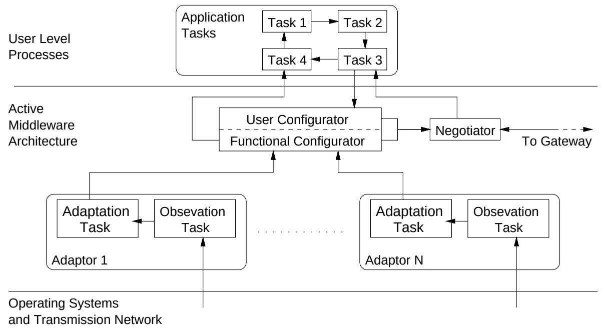


Figure 2. Client Architecture

The client middleware assists the application with the provision of best QoS possible, using the underlying best-effort resource management. The central component is the configurator. The configurator consists of two components: the *functional configurator* and the *user configurator*. The functional configurator performs coarse application-level reconfigurations using a fuzzy inference engine when the fine-grain adaptations performed by the adaptors are insufficient for adapting to the current observed resource states. The user configurator performs high-level reconfigurations initiated by user responses to Quality of Perception (QoP) degradation. QoP degradations occur when application-specific quality (e.g. view of a physical object) is degraded independently of the underlying resource availability [3]. For example, within the ODVC system, a functional reconfiguration may switch compression schemes for the video data, whereas a user reconfiguration may switch to a server with a better view of the scene.

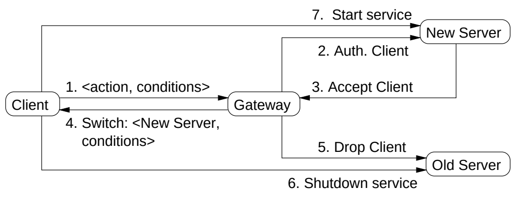


Figure 3. The Server Switching Protocol

When a reconfiguration requires switching servers, the *negotiator* is used to negotiate a new server from the gateway. The protocol between the client's negotiator and the gateway is shown in Figure 3. In step 1, the negotiator sends the gateway the action it needs to perform and a list of conditions the action must satisfy. The gateway then ranks the servers using a fuzzy inference engine identical to the configurator's. It then cycles through steps 2 and 3 in decreasing order from the top ranked server until a server accepts the client. When the gateway returns the selected server in step 4, it must also provide a list of conditions the server satisfies so that the client application can adjust itself to the conditions met by the server. For example, the client's action could be to switch to a new format and the conditions could include a list of acceptable formats. The gateway would then return not only the new server but also the specific format from the list which the server satisfies.

2.4. The Gateway

The gateway serves as the centerpiece in the connection between each client and the facility. It is responsible for receiving client reconfiguration requests and processing the requests for the client.

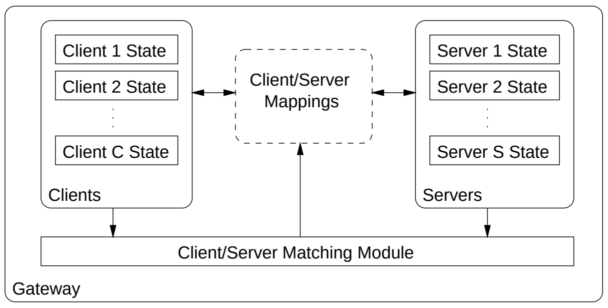


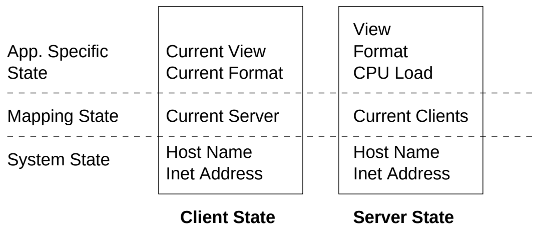
Figure 4. Gateway Architecture

The gateway's architecture can be seen in Figure 4. The gateway maintains two state tables, one for the clients and one for the servers. An example of gateway states is shown in Figure 5. Between these is a mapping table which maintains which clients are being serviced by which servers. This table is updated by the *client/server matching module*.

The matching module is invoked during the server switching protocol described in section 2.3. This module is responsible for assigning a rank to each server. The rank is computed using a preprocessed representation of the action, conditions, client state, and server state. This representation is inputted into the fuzzy inference engine with a rule base specifically created for the gateway. For example, a rule for moving left (clockwise) within an ODVC is as follows:

```
if (server_direction is left) and (serv-
er_angle is close) then server_ranking is
high;
```

In this example, *server_direction* and *server_angle* are preprocessed values derived from the views on the client's current server and the evaluated server. The better a server matches the criteria for the action, the higher the server's ranking will be. Additional information can be encoded into each rule to create a more robust rule base.



	Client State	Server State
App. Specific State	Current View Current Format	View Format CPU Load
Mapping State	Current Server	Current Clients
System State	Host Name Inet Address	Host Name Inet Address

Figure 5. ODVC Gateway States

3. Omni-Directional Video Camera

The ODVC is a suitable experimental application because it provides for demonstrating the capabilities of the framework.

The ODVC's client has two reconfiguration actions the gateway can satisfy: requesting compression and requesting camera movement. Upon evaluating a client's request, the gateway's rule base ranks the servers on four criteria: the video format and CPU load on the new server, and the direction and distance within the ODVC between the new server and the client's current server. The rules are written so that different rankings apply given different actions. For example, a format request does not take into account the server's direction while a move request assigns a very low ranking to a server in the wrong direction.

4. Implementation

To test our framework, we implemented it in two steps. In the first step, we created the framework itself. In the second step, we extended the framework towards an ODVC as a component of our OmniTrack visual tracking application [6]. Our implementation was created for Windows NT 4.0 SP5 using Visual C++ 6.0. We used CORBA interfaces to communicate between the various middleware components.

5. Experiments

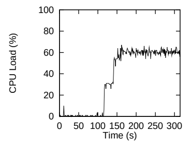
We have designed three experiments to demonstrate the gateway's ability to provide each client with a QoS-satisfactory server. We evaluate the gateway's server decision using two criteria: how well the selected server matched the client's reconfiguration request, and how well the gateway performed load balancing while satisfying these requests.

Each experiment was performed using six clients and three servers. The clients were placed on hosts of varying speed, one client per host. The servers were placed on three hosts, one server per host, in descending order or host processor speed: an MJPEG server on a Pentium II 400 (PII 400), another MJPEG server on a dual processor Pentium Pro 200 (PPro 200), and an uncompressed server on a Pentium 200 MMX (P 200). The gateway was placed on "PII 400". For all three experiments, each server could satisfy each client's initial QoS parameters: any video format with any view.

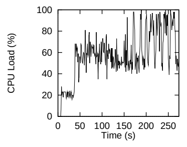
For the first experiment, we manually selected the server for each client to best balance the overall server load. This experiment was performed as a control for evaluating the second and third experiments. The results of this experiment are shown in (a, b, c) of Figure 6. The jumps in execution levels indicate when each server begins servicing another client. Notice that although server "P 200" reaches 100% utilization, at the time the second client was placed on the host it had the lowest utilization of all three servers.

For the second experiment, we repeated the first experiment, this time allowing the gateway to select the server for each client. The results of this experiment are shown in (d, e, f) of Figure 6. As expected, the gateway satisfied all client requests while performing an equivalent level of load balancing which had been performed manually in the first experiment.

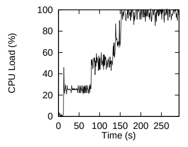
In the second experiment, the relative load on each server was close enough such that a random or round robin client placement would have performed adequate load balancing. To prove that the gateway would respond equally well within a more biased service facility, we switched the fastest server, "PII 400", from MJPEG to uncompressed video. This change greatly reduced the processor overhead for this server. For the third experiment, we allowed the gateway to select servers with the new configuration. This time, the gateway placed every client on "PII 400". Even with the gateway executing and a server servicing all six clients, the CPU load on this host was still lower than the load on the other non-servicing hosts. Thus, the gateway successfully demonstrated that it could meet a client's QoS needs while maintaining optimal utilization across the facility.



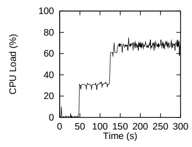
(a) PII 400



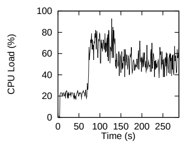
(b) PPro 200



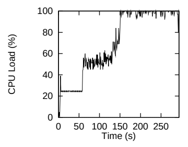
(c) P 200



(d) PII 400



(e) PPro 200



(f) P 200

Figure 6. Experiment CPU Loads

6. Conclusions

In this paper we presented a novel approach for performing high level QoS adaptations for a distributed application. Our approach does this by using several servers and a gateway working together to provide the client with flexible QoS reconfiguration options via the set of available servers. We then demonstrated one extension of our framework, the omni-directional camera.

References

[1] E. Amir, S. McCanne, and H. Zhang. An Application Level Video Gateway. *Proceedings of ACM Multimedia '95*, 1995.

[2] S. Brandt, G. Nutt, T. Berk, and M. Humphrey. Soft Real-Time Application Extension with Dynamic QoS Assurance. *IWQoS 98*, 1998.

[3] S. Fish, G. Ghinea, and J. P. Thomas. Mapping Quality of Perception to Quality of Service for a Runtime-adaptable, Communication System. *SPIE MMCN '99*, 1999.

[4] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. *IEEE Personal Communications, Special Issue on Adaptation*, 1998.

[5] M. Garland, S. Grassia, and S. Puri. Implementing Distributed Server Groups for the World Wide Web. *Carnegie Mellon University School of Computer Science Technical Report CMU-CS-95-114*, 1995.

[6] B. Li and K. Nahrstedt. A Control-based Middleware Framework for QoS Adaptation. *IEEE Journal of Selected Areas in Communications*, Sept. 1999.

[7] D. Xu, D. Wichadakul, and K. Nahrstedt. Multimedia Service Configuration and Reservation in Heterogeneous Environments. *ICDCS 2000*, 2000.