Impact of Control Theory on QoS Adaptation in Distributed Middleware Systems

Baochun Li
Electrical and Computer Engineering
University of Toronto
bli@eecg.toronto.edu

Klara Nahrstedt *
Department of Computer Science
University of Illinois at Urbana-Champaign
klara@cs.uiuc.edu

1 Introduction

In ubiquitous computing environments, various distributed multimedia applications will be desired. Examples of such applications are video-on-demand, video conferencing, visual tracking and others. For these applications to be accepted, they must deliver desired quality of services (QoS) to the users such as desired video frame rate, tracking precision, and minimal loss of information. This is especially challenging in shared computing and communication environments with no underlying operating and networking system support for QoS. The difficulty lies in fluctuation and unpredictability in resource availability when multiple tasks share processor, network bandwidth or memory. However, cost of computing and communication infrastructure for distributed multimedia applications forces the application designers to share the infrastructure, hence *distributed middleware systems* are needed to help applications with underlying fluctuations in resource availability. Middleware systems reside between the operating system and the applications and one of the key roles of middleware systems is to assist applications in their goal of delivering the required QoS. The approach, used in middleware systems, is *QoS adaptation*.

There exist many heuristic algorithms to do QoS adaptation utilizing buffering approaches [1], smoothing algorithms [2], acknowledgment-based approaches [3], priority-based algorithms [4] and others. All these algorithms present possible system mechanisms, protocols, policies, hence partial solutions regarding how to deal with QoS adaptation, but they do not present a viable model and framework for QoS adaptation which allows (1) to reason about QoS adaptation properties such as stability, agility and fairness, and (2) to introduce new adaptation algorithms in an integrated and scalable fashion.

In this extended abstract we outline an integrated and scalable control model for QoS adaptation (Section 2), and show how it can benefit the quality control in distributed middleware systems (Section 3). We have built a prototype of a middleware system, called *Agilos*, which verifies our control model and shows the soundness of our approach (Section 4).

2 Control Model for QoS Adaptation

The major objective of a middleware system is to control an application task so that critical quality parameters are delivered. In order to control an application, we need to establish a *task control model*. Our control task model consists of the *target task* to be controlled, the *adaptation task* that implements the control algorithm, and the *observation task* that observes task states in the target task (Figure 1).

The target task, utilizing the control theory, can have the following form:

$$\frac{d\mathbf{x}(t)}{dt} \equiv \dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t), t] \tag{1}$$

$$\mathbf{z}(t) = \mathbf{h}[\mathbf{x}(t), \mathbf{v}(t), t] \tag{2}$$

With the above definition, the target task is said to be at *equilibrium* when:

* This work was partially supported by the Air Force grant under contract number F30602-97-2-0121 and National Science Foundation Career grant under contract number NSF CCR 96-23867.

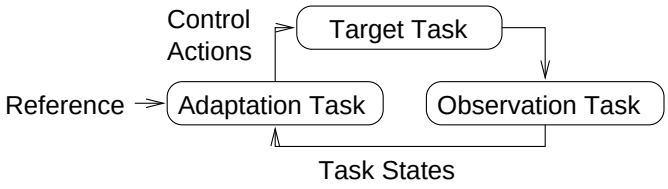


Figure 1. The Task Control Model

$$\dot{\mathbf{x}}(t) = 0 = \mathbf{f}[\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t), t] \tag{3}$$

where $\mathbf{x}$ denotes the vector of task states, $\mathbf{u}$ the vector of controllable input parameters, $\mathbf{z}$ the vector of observed output parameters of the task, $\mathbf{w}$ the uncontrollable variations in the task, and $\mathbf{v}$ the observation errors.

The above stated definitions are generic and may be non-linear, time-varying, and too complex to derive a suitable control algorithm. Hence, in order to speed up and simplify the control, we will approximate the task control path by piecewise linear functions and work with target task model as follows:

$$\mathbf{x}(k) = \mathbf{\Phi}\mathbf{x}(k-1) + \mathbf{\Gamma}\mathbf{u}(k-1) + \mathbf{w}(k-1) \tag{4}$$

$$\mathbf{y}(k) = \mathbf{H}\mathbf{x}(k) \tag{5}$$

$$\mathbf{z}(k) = \mathbf{y}(k) + \mathbf{v}(k) \tag{6}$$

We have derived a concrete model of a target task based on the above simplified generic model, in the scenario shown in Figure 2.

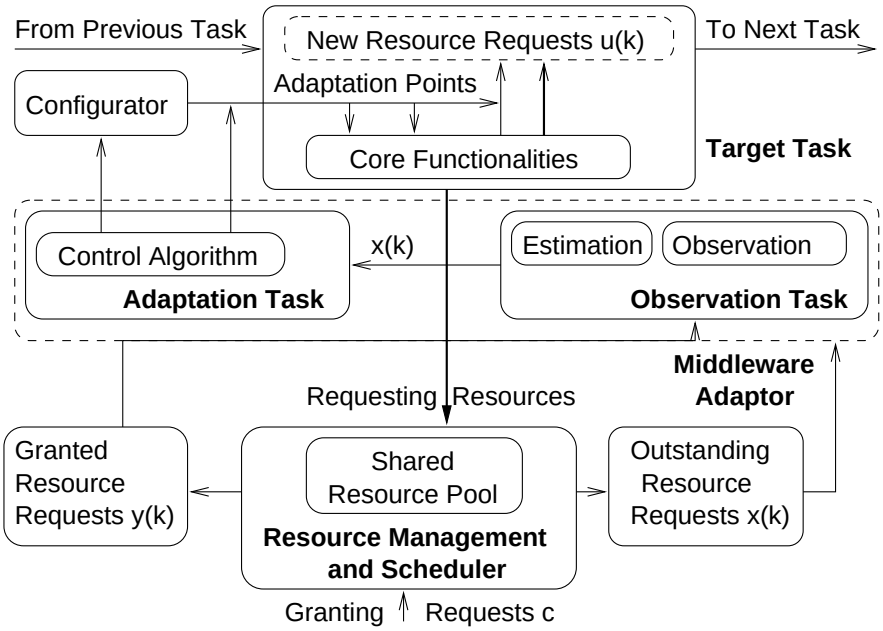


Figure 2. A Concrete Model for the Target Task

The model assumes a resource pool which is shared among multiple target tasks, and each application task makes requests for resources through its middleware system. These requests are either granted or outstanding. The system is granting c requests within a given time interval for multiple running tasks. If one target task has a critical QoS parameter which requires a threshold, e.g., u_s requests to be satisfied, then the observation and adaptation tasks of the task control model, residing in the middleware, must provide control actions to adapt and adjust resource requests of other target tasks.

Equation (7) provides an analytical model for the above observations:

$$\frac{x(k)-x(k-1)}{k-(k-1)} = x(k)-x(k-1) = \sum_{i=0}^{M(k-1)} u_i(k-1) - c \quad (7)$$

where $x(k)$ is the total number of *outstanding resource requests* made by all concurrent tasks at time k, $u(k)$ is the controlled request rate by the controller (the adaptation task), and $M(k)$ is the total number of active concurrent application tasks. c is the request granting rate. Intuitively, the model captures the general fact that the difference between resource requesting rate under control (for all active tasks) and resource granting rate is equivalent to the changes in total number of outstanding resource requests.

Our control model assumes that application target tasks have a good understanding of their critical QoS parameters and their mappings to thresholds in order to trigger appropriate adaptive control actions. In order to learn about appropriate thresholds, we have introduced a novel middleware component *QualProbes* to *probe* the dynamic behavior of the application task at run-time during experimental runs, and then apply the result of probing to determine related thresholds, which are applied during actual field runs.

Using our concrete task control model, we are able to show [5] stability and agility properties for QoS adaptation. For shared resources (e.g. CPU) we can show the fairness property as well.

3 Adaptive QoS Control in Distributed Middleware Systems

Our concrete task control model for QoS adaptations must withstand not only minor fluctuations in shared resource availability, but also large changes. For this larger scale of fluctuation, it is necessary that the adaptation task of the task control model be designed in a hierarchical fashion. The hierarchical design ensures that small changes in resource availability can be controlled by adaptation tasks, acting as linear controllers (e.g. a PID controller) to adjust the request rate, resulting in small QoS changes. On the other hand, large changes in resource availability require very often functional changes in the target task, hence these types of changes can be controlled by fuzzy controllers, which we refer as *configurators*. Along with observation tasks, the fuzzy and linear controllers are integrated to form a hybrid control mechanism, shown in the feedback loop in Figure 3.

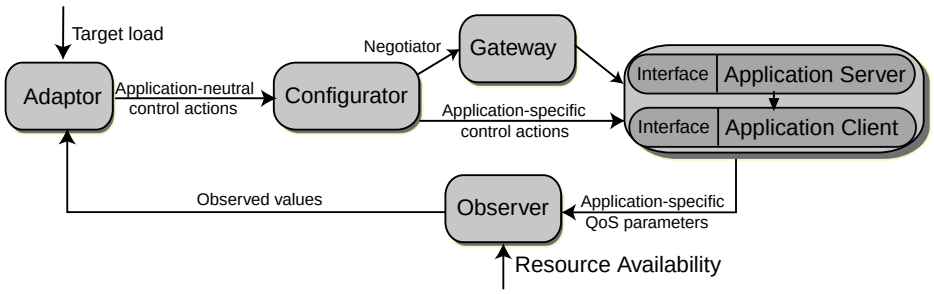


Figure 3. The Control Loop In the *Agilos* Architecture

When integrated with other value-added services such as QualProbes, negotiators and gateway services [6], we have designed *Agilos*, a hierarchical control-based middleware framework to support application-level QoS adaptations. The overall architecture of *Agilos* is shown in Figure 4. In the figure, *adaptors, configurators, and negotiators* implement the adaptation tasks in the task control model, and *observers* implement the observation tasks. As our experiments in Section 4 will show, *Agilos* works across end hosts in a distributed fashion, effectively controlling the application to achieve a better performance via adaptations.

4 Validation with OmniTrack

We have implemented the *Agilos* framework under Windows NT, and have carried out a series of experiments with *OmniTrack*, a distributed omni-directional tracking application, which have effectively validated the design of the *Agilos* framework. Servers in *OmniTrack* are able to serve live video streams to clients, which execute visual tracking algorithms that track the objects of interest. Our focus is on the *critical QoS parameter*, which is the tracking precision for the case of *OmniTrack*.

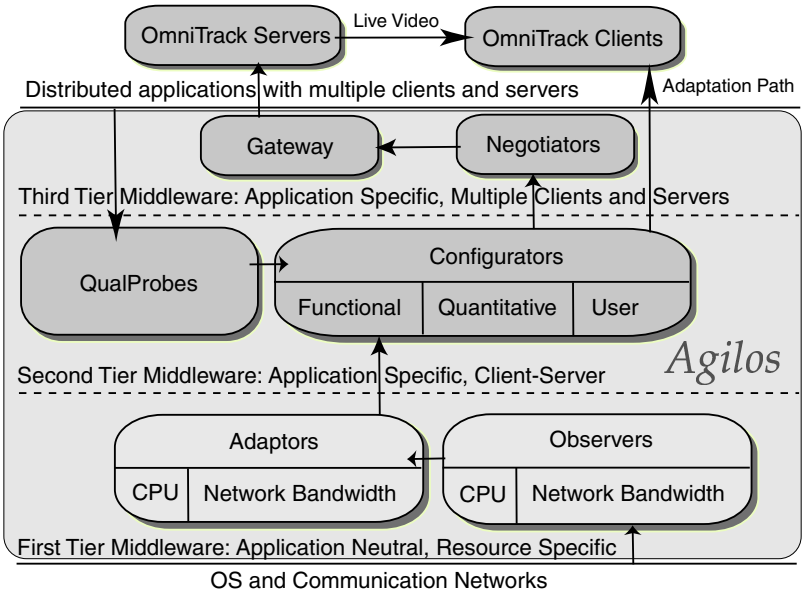


Figure 4. The Hierarchical Design of the *Agilos* Architecture

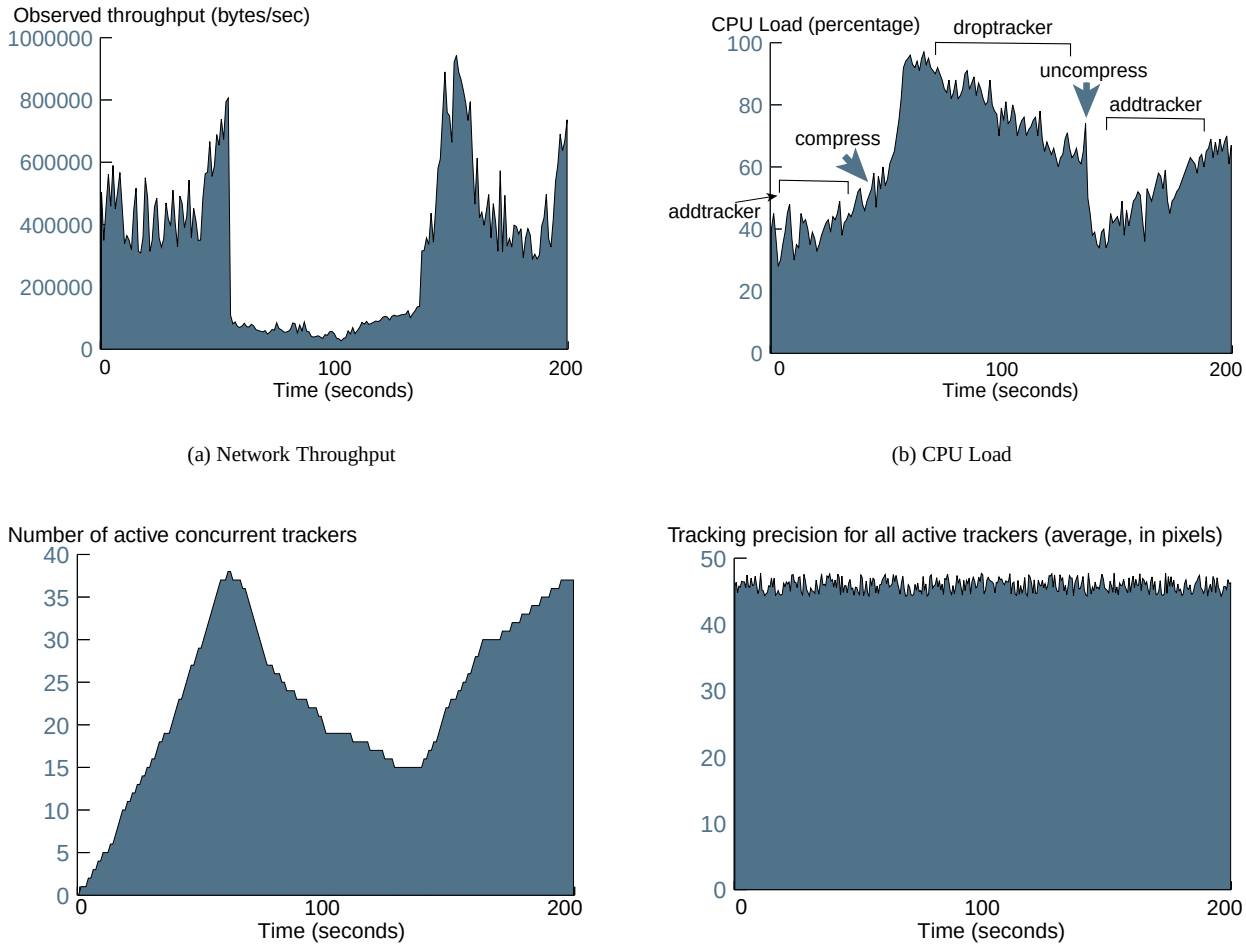
Figure 5 shows the results we have obtained, in one of the experimental scenarios. Table 1 shows the control actions generated by the configurator at their respective starting times. The timing of these control actions is also visually embedded in Figure 5(b).

Start Time (sec)	*Control Action from Configurator*
4-40	`addtracker`
56	`compress`
62-120	`droptracker`
139	`uncompress`
145-178	`addtracker`

Table 1. Control Actions generated by the Configurator

In Figures 5(a) and 5(b), we show observations with respect to network throughput and end-system CPU load. These observations drive the first tier adaptors, whose output drives the behavior of functional configurators in the second tier. In the case of this scenario, a hybrid combination of parameter-tuning and reconfiguration choices is activated by *Agilos*. The specific reconfiguration choices being used are `compress` and `uncompress`, which activates and deactivates the Motion JPEG video codec in both the tracking server and client. The tunable parameter is the number of active trackers. Since the trackers are executed in a round-robin fashion, the duration of executing all trackers in each iteration is increased when there are more active trackers present. This may impose a heavier CPU load on the end system. Figure 5(c) shows the number of active trackers, which is the result of the control actions `droptracker` and `addtracker` generated by the configurator. The reconfigurations `compress` and `uncompress` significantly reduces the network throughput from the server to the client, and increases the CPU load to near 100%. This leads to another round of parameter tuning adaptations related to `droptracker`. As illustrated in Figure 5(d), this combination of parameter-tuning and reconfiguration choices, activated with appropriate timing determined by the configurator, leads to a stable collective tracking precision for all trackers. Such measurement of tracking precision is obtained by calculating the average of the precision of individual trackers.

Based on the above analysis, we conclude that a hybrid control framework, consisting of a linear PID controller and a fuzzy controller, is effective for maintaining the stability of the critical QoS parameter, and presents a feasible solution for QoS adaptation in distributed middleware systems.



(a) Network Throughput

(b) CPU Load

(c) Number of Active Concurrent Trackers

(d) Average Tracking Precision for All Trackers

Figure 5. Experimental Results for Scenario 2

References

[1] M. Shor, K. Li, and J. Walpole, "Application of Control Theory to Modeling and Analysis of Computer Systems," in *Proceedings of Japan-USA-Vietnam Workshop on Research and Education in Systems*, 2000.

[2] G. Cao, W. Feng, and M. Singhal, "Online VBR Video Traffic Smoothing," in *Proceedings of 8th IEEE International Conference on Computer Communications and Networks*, 1999, pp. 502–507.

[3] Z. Chen, S. Tan, R. Campbell, and Y. Li, "Real Time Video and Audio in the World Wide Web," in *Proceedings of Fourth International World Wide Web Conference*, 1995.

[4] H. Chu and K. Nahrstedt, "CPU Service Classes for Multimedia Applications," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, 1999.

[5] B. Li and K. Nahrstedt, "A Control-based Middleware Framework for Quality of Service Adaptations," *IEEE Journal of Selected Areas in Communications*, vol. 17, no. 9, pp. 1632–1650, Sept. 1999.

[6] W. Kalter, B. Li, W. Jeon, K. Nahrstedt, and J. Seo, "A Gateway-Assisted Approach Toward QoS Adaptations," in *Proceedings of IEEE International Conference on Multimedia and Expo*, Aug. 2000, pp. 855–858.