InfoFilter: Supporting Quality of Service for Fresh Information Delivery

Ling Liu, Calton Pu, Karsten Schwan, Jonathan Walpole*

Georgia Institute of Technology, Atlanta, GA 30332-0280, USA
** Oregon Graduate Institute, Portland, OR 97291-1000, USA*
{lingliu,calton,schwan}@cc.gatech.edu, walpole@cse.ogi.edu

Received February 4, 2000

Abstract With the explosive growth of the Internet and World Wide Web comes a dramatic increase in the number of users that compete for the shared resources of distributed system environments. Most implementations of application servers and distributed search software do not distinguish among requests to different web pages. This has the implication that the behavior of application servers is quite unpredictable. Applications that require timely delivery of fresh information consequently suffer the most in such competitive environments. This paper presents a model of quality of service (QoS) and the design of a QoS-enabled information delivery system that implements such a QoS model. The goal of this development is two-fold. On one hand, we want to enable users or applications to specify the desired quality of service requirements for their requests so that application-aware QoS adaptation is supported throughout the Web query and search processing. On the other hand, we want to enable an application server to customize how it should respond to external requests by setting priorities among query requests and allocating server resources using adaptive QoS control mechanisms. We introduce the Infopipe approach as the systems support architecture and underlying technology for building a QoS-enabled distributed system for fresh information delivery.

Keywords Distributed Information Flow Systems, Web Information Systems, Quality of Service, Adaptive Resource Management.

§1 Introduction

On the Internet, users issuing search queries to remote information servers often experience large variations in important performance metrics and information qual-

ity metrics. Typical performance variations include data transfer bandwidth and access delay. Typical information-quality (IQ) variations include the amount of false positives (useless answers that fail to fulfill a user's needs) and false negatives (useful answers that the system fails to deliver to the user) in the search results, the freshness and coverage of the information delivered, and the information representational consistency. These variations are primarily caused by the wide range of server capabilities, such as time-of-the-day differences or server-dependent differences in network paths, network load, server-specific query capabilities, and server utilization. Furthermore, most implementations of application servers and distributed search software treat all requests uniformly. This has the implication that the behavior of application servers is quite unpredictable as analyzed in [6]. First, requests for popular pages have the tendency to overwhelm the requests for other (and possibly more time-sensitive) pages. Second, pending requests may completely bog down the servers, resulting in unacceptable response time. Third, servers may start to drop requests indiscriminately. Fourth, servers may deliver out of date results. It is becoming increasingly important for distributed systems to be able to handle application demands for resources more intelligently.

In this paper, we present the initial results of our research towards developing an application-aware quality of service (QoS) framework for managing distributed systems resources in order to provide application- level QoS guarantees, and ultimately supporting smart delivery of fresh information. This research consists of three main components. The first component is a model of QoS for fresh information delivery. The second component is the Infopipe approach to designing and implementing a QoS guaranteed Web information delivery system – InfoFilter. The Infopipe approach is the core technology of the Infosphere Project [7], one of the five pioneer projects under the DARPA ITO Information Expedition program. The third component is an adaptive, micro-feedback driven approach to distributed systems resource management that dynamically configure the available resources in terms of the QoS demands of applications. This paper presents the QoS model in the context of Web query and search processing and briefly describes the InfoFilter system that enforces the proposed quality of service model using Infopipes technology.

The model of quality of service (QoS) presented in the paper aims at enabling an application server to customize how it should respond to external requests. This includes setting priorities among page requests, allocating different kinds of (absolute and relative) service resources to different requests. More concretely, it allows the QoS parameters to be expressed in terms of different units

of work and different layers of the system. Built on top of the taxonomy of QoS specification developed in [3], we explicitly distinguish application-level QoS parameters (such as frame delay and frame jitter for a video) from resource-level and system-level QoS parameters (such as the packet delay and packet jitter at network resource level). We model application-level QoS parameters as a function of the appliation's goals specified by application designers. We describe resource-level QoS parameters in terms of the design of the resources and their application-aware control parameters.

To implement such a QoS model, first, the system needs to provide an application-level QoS specification language to allow users to specify the desirable quality control parameters with their queries. Second, mechanisms are needed to translate the application-level QoS parameters into resource-level and system-level QoS parameters. Third and most importantly, an implementation requires the creation of a resource model for determining various resources that exist at any given moment. This paper describes an adaptive resource management mechanism for scheduling various requests given a resource model such that the QoS constraints are satisfied. A key building block of the InfoFilter QoS system is the use of infopipes. The Infopipe approach provides a viable and effective technology to support distributed information flows with QoS requirements. It includes the flexible composition of Infopipes while preserving QoS properties, which is critical for implementing a QoS-aware distributed system for timely delivery of fresh information.

§2 The InfoFilter Quality of Service Model

Much of the quality of service management research results have been produced and published in the context of networking and multimedia systems where resource consumption needs often exceed the available resource capacity of a system [10, 12, 8], or where resources are allocated unfairly during periods of network congestion. In a consumer-producer framework, quality of service can be seen as a quantification of level of services that an information production server can guarantee its consumers. Often, the selection of quality of service parameters depends on the kind of services that a server provides. Examples of typical parameters that multimedia systems have used to guarantee services are transmission delay, network transfer rate, image resolution, video frame rate, and audio or video sequence skew, among others. In this section, we develop a model of quality of service for distributed information servers, such as Web application servers, continual query servers [4], and geographic information servers.

The design of the InfoFilter QoS model follows a number of design decisions. The first design consideration is to make an explicit distinction between two views of the quality of service: consumer view and producer view. In the consumer view, the application server guarantees specific services to its consumers according to the desirable level of services specified by consumers. Examples of such QoS guarantees are a server's resource guarantee for lower bounds on its throughput (e.g., number of bytes per second) or upper bounds on response times for specific queries. In the producer view, the quality of service implements the producer's view of how the producer's server should provide certain services, including policies for setting priorities among various resources and setting limits on server resource usage by various types of requests.

The second design consideration is the need for establishing a common understanding of how QoS should be specified at application level, resource level and system level [3]. In order to design a system that allows multiple applications to co-exist within a QoS management framework, it is necessary to define a common and coherent QoS model. Such a QoS model should not only incorporate various individual specifications from consumers but also be able to map QoS properties from application level to resource level and from resource level to system level.

2.1 QoS Definition

In InfoFilter, an application is modeled as an information flow system using a directed graph, where graph nodes represent processes and graph edges represent information flow between those processes. A process can be either atomic or composite. We use atomic processes to denote units of work and composite processes to refer to services or composition of services (i.e., a service may use other services to complete a task). A unit of work represents the smallest granularity of work for which only a single resource needs to be allocated [3]. A service is defined as a collection of one or more units of work that may span multiple resources. Therefore, a unit of work can also be referred to as a service.

We define end-to-end QoS requirements (parameters) for each service and describe the resource usage as a function of the QoS. Thus, a single QoS specification is provided for the entire service. The consumer of a service negotiates the QoS of the entire service without having to understand the units of work that made up the service.

The InfoFilter application QoS model is developed as an extension to the Quasar QoS model [11], which was designed specifically for multimedia systems and video-on-demand in particular [11]. Similar to the Quasar QoS model, we model

the quality of a query result as a measure of the amount of error present in it. For example, a query returning a perfect result would return an error-free replica of the real world object(s). We explicitly distinguish capture error, quantization error (sampling error), and delivery error. The *capture error* refers to the class of errors that result from the use of inaccurate capture equipment or less than perfect information extraction software. These errors are considered as incidental because they may not be present when different capture (software or hardware) or data extraction tools are used. The *quantization error* represents the class of inherent errors that result from the use of a finite number of samples (and a finite number of bits per sample) to represent time varying values from the real world. The *delivery* error describes the class of errors introduced by resource management decisions that influence the processing and the delivery of query results. Delivery errors in Web query systems are primarily caused by page and packet oriented data transfer delays, buffering delays, the choices of resource scheduling policies, and the unexpected server unavailability problems. Below we discuss a list of quality of service parameters that are common for Web application servers and describe consumer requests with QoS specifications.

2.2 QoS Specification

We define the overall quality of a Web query request as the degree of user satisfaction with the query results and the delivery efficiency according to the user's QoS requirements. While user satisfaction is qualitative and subjective, the delivery efficiency can be measured against the QoS specifications. We introduce a set of parameters to be used for incorporating quality of service control into the construction, operation, and maintenance of consumer requests.

Table 1 lists a subset of QoS parameters to be used in the InfoFilter consumer request construction. If we consider the QoS specification as a performance metric, then the first four parameters in Tabel 1 are the timeliness parameters. The deadline for the query to complete measures the time affordable by the consumer to wait for the query to return the results. The total time taken to complete a query measures the query round trip time – from the time a query is submitted to the time the execution of the query is completed (i.e., all results are returned). The rest of parameters are either the precision parameters with respect to the volume of the data flows or the accuracy parameters that measure the errors introduced into the query results.

parameter	Synopsis	Example
DL	The deadline for the query to complete	30 seconds
RT	Query response time – from the time a query is submitted to the time the first piece of data appears	20 seconds
TT	The total time taken to complete a query	1 minute
JT	Jitter – The variability in time to complete the query (measuring the internal consistency of timeliness parameters)	0.5 seconds
QS	Query scope – query search scope in a Web document	`<B>,<HREF>`
QC	Query coverage – Number of network nodes or Web sites accessed	250 sites
FR	Freshness of query result – the usefulness duration of a result item since its last modification	5 days
RD	Redundancy rate of query result – percentage of duplicate items	0.004
AC	Accuracy of the query – the percentage of the retrieved documents or result items satisfying the query condition	0.9
PR	Precision of the query – the fraction of the retrieved documents or result items which is relevant	0.99
RC	Recall of the query – the fraction of relevant documents or result items that has been retrieved	0.12
RL	Relevance of the query result – percentage of result items that may not satisfy the query condition	0.15
ND	Number of Web documents (files) accessed	10,000 docs
NA	Total size of the query result – the number of result items returned in an execution of a query	320 items

Table 1: Typical QoS parameters for Web queries

The term "result item" used in Table 1 is defined as a data object or an URL of a related document for Web information sources. The query scope (QS) is defined in terms of a subset of record field tags for data files or a subset of HTML (or XML) tags for HTML (or XML) documents. A QS value specifies a minimum set of content tags that the InfoFilter query processor has to search. For example, a QS value {`<H2>,<B>,<HREF>`} means to search at least the text appearing in the header parts, the bold parts, and the embedded URL links of each HTML document. A QS value `Table` means to search the table defined by the pair of start and end table tags `<table>` and `</table>`. The default value of QS is `<HTML>`, which means to search the whole document. A freshness value FR indicates the duration of an item since its existence. It is an important quality measurement for Web queries since most of the users are not be interested in "out of date" information. The accuracy of the query result (AC) is defined as

$$AC = NC/NA$$

where NC is the number of correct items in the query result. We define the precision PR of a query as follows:

$$PR = (NA \cap NR)/NA$$

where NR is the total number of relevant documents or relevant objects at the sources of the query. $(NA \cap NR)$ is the total number of retrieved documents that is relevant. A precision of 90% means that 90% of the query result are relevant and there are 10% of irrelevant result items in the result. We define the recall of a query as

$$RE = (NA \cap NR)/NR.$$

A recall of 100% means that the query returns all the relevant objects at the sources. A recall of 90% means that the query returns 90% of the relevant objects and missed 10% of the relevant objects.

Typically, users specify a requirement threshold (minimum/maximum value) for each quality parameter when he/she issues a request. The InfoFilter request manager checks if the execution of the query meets the quality requirements by examining the values of quality parameters. The quality of service inspection module sets off an alarm when a quality parameter drops below the minimum required QoS value or rises above the maximum. We refer to the QoS values specified by a user as *user-defined* quality parameter values. In contrast, the quality parameter values obtained at run time during Web query processing are called *execution-time* quality parameter values. We say that a Web query execution is *successful* under QoS control if all of its execution-time parameter values are equal to or better than (smaller or greater depending on the semantics of each parameter) the user-defined quality parameter values.

2.3 QoS as a Distance Measure

A distance function [11] is defined over a single parameter of a given QoS metrics. It is used to measure the distance between two quality parameter values. Such distance value is a useful indicator of the relative goodness of the two QoS parameter values.

Distance function.

Let P be a QoS parameter, $domain(P)$ denote the domain of P, and $domain(P) \neq \emptyset$. Let $u, v, w \in domain(P)$. A function $\delta_P(u, v)$ is a distance function for the parameter P, if it has the following properties:

1. δ_P is a function from $domain(P) \times domain(P)$ to a set of real numbers, denoted by R, and $domain(R) \stackrel{\text{def}}{=} [0, 1]$;
2. $\forall u, v \in domain(P)$, $\delta_P(u, v) = \delta_P(v, u)$; (Symmetry)
3. $\forall u, v, w \in domain(P)$, $\delta_P(u, v) + \delta_P(v, w) \geq \delta_P(u, w)$. (Triangle inequal-

ity)

Consequently $domain(P)$ is a metric space. Let each value in $domain(P)$ denote a presentation state of the query result object o. The distance function δ_P can be defined as the absolute value of the difference between two presentation states of the object o. By the definition of $\delta_P(u, v)$ where $u, v \in domain(P)$, the following properties holds:

(a) $\delta_P(u, v) = 0$ if and only if $u = v$.

(b) $\delta_P(u, v) > 0$ if and only if $\delta_P(v, u) < 0$.

(c) if $\delta_P(u, v) > 0$ and $\delta_P(v, w) > 0$, then $\delta_P(u, w) > 0$.

The property (a) amounts to say that two property values are identical if their distance is zero. The property (b) implies that if a property value u is better than v, then v is worse than u. The property (c) says that if u is better than v and v is again better than w, then we can say u is better than w. These properties are frequently used in QoS control systems.

Consider the list of QoS properties in Table 1. For response time RT, round trip time TT, freshness FR, redundancy rate RD, and relevance RL, the smaller the parameter value is, the higher the quality. Therefore, we define the distance function as

$$\delta_P(u, v) = v - u, \text{ where } P \in \{RT, TT, JT, FR, RD, RL\}.$$

However, for other parameters such as accuracy AC, precision PR, recall RE, the larger the parameter value is, the higher the quality. We define the distance function as

$$\delta_P(u, v) = u - v, \text{ where } P \in \{AC, PR, RE\}.$$

For the rest of parameters the definition of its distance function is more sophisticated. User-defined quality criteria can be used in such cases to determine the best quality between two sets of parameter values. For instance, a user may consider that the query results returned by searching over a larger coverage are better in quality.

Very often, users need to specify a set of quality parameters for each consumer request. To compare two sets of quality parameter values, we need to introduce an aggregate distance function that allows us to compute the distance between two multi-dimensional quality parameter vectors. The design of a concrete distance function is a subject of our ongoing research. One possible solution is to define the

aggregate distance function as a weight function by assigning a weight to each quality dimension. The sampling approach or self-adaptive learning approach can be employed to determine the weight values for different quality dimensions.

2.4 Constructing Requests with QoS Parameters

In InfoFilter, two types of consumer requests are supported with quality of service guarantee. The first type of requests is called ad-hoc queries. An ad-hoc query performs a conditional search over the remote information sources. The second type of requests is called QoS-guaranteed query subscription. Each query subscription is modeled as a continual query[*1]. We define a query subscription in terms of four components: triggering event, standing query, start and stop condition. The triggering event can be a recurring time event (e.g., every 10 minutes), a system state (e.g., when a thermometer reaches the temperature of 100F), or a combination of both. The standing query is a normal query on the data sources (in this case, pulling selected sensor data) that is executed each time the triggering event becomes true. The query result is pushed to the user or program that created the request whenever the trigger condition is met. Like the triggering event, the start and stop condition of a query subscription can be a combination of time-based or content-based events. A subscription is deactivated after the stop condition has occurred. For example, the query subscription "*transmit the last 2 minutes of buffered infrared videotape when the seismic sensor indicates an explosion nearby*" can be specified as an InfoFilter continual query.

Formally, a QoS guaranteed query subscription, denoted as (f_{cq}, f_{QoS}), is defined by a continual query component f_{cq} and a QoS specification component f_{QoS}. We define the continual query component f_{cq} as a quadruplet (Q, T_{cq},`Start`,`Stop`), consisting of a normal query Q, a trigger condition T_{cq}, a begin condition `Start`, and a termination condition `Stop`. T_{cq}, `Start`, and `Stop` in general may depend on many different parameters, and in the sequel we omit their parameters for clarity. In contrast to ad-hoc queries in conventional database systems or current Web search engine-based information retrieval systems, an InfoFilter query subscription, once activated (installed and started) runs continually over the set of information sources. Whenever its trigger condition becomes true, the new result since the previous execution of the query will be returned if it meets the QoS specification. Below we illustrate the construction of InfoFilter query subscription requests using

[*1] A continual query (CQ)[4] is a standing query that, once installed, runs continually over the targeted information sources and returns the new results when the amount of information updates reaches a specified threshold.

an example:

Example 2.1
"I want all SAR (Synthetic Aperture Radar) Imagery within 100 miles of my (time-varying) location". Suppose this query has the following quality requirements: the precision of 100 miles of my location is 90%, the duplication rate of the query result is zero, and the maximum query turnaround time latency is 2 minute. We can perform this request by installing the following continual query:

f_{cq} = (Q : `Select * From SARImagerySource`
`Where my_location - 100 miles < distance_range`
`AND distance_range < my_location + 100miles`,
T_{cq}: `my_location changes From LocationCoordinateSource`,
`Start`: *now*,
`Stop`: *24 hours*)
f_{QoS} = (`TT:2min,SC:{<SARImagery>,<distance_range>,<LocationCoordinate>}`,
`AC:0.9, RD:0.0`).

Note that we use the trigger condition T_{cq} to monitor the changes of "my_location" and use the query Q to filter those SAR Imagery within 100 miles of distance to each given value of "my_location". The QoS specification of the request will be translated into lower level of QoS properties for fine-granularity resource management and control by the dynamic QoS adaptation controller.

2.5 QoS Specification of Server Resources

It is well understood that a QoS model must tie the users' needs (application QoS) to the amount of resources required to provide them. The resource-level QoS specification describes the system resources that are required by the server to fulfill the application requests. Typical server resources associated with consumers' requests to Web pages include *resource types* (such as the percentage of server resources allocated to a page request), *performance characteristics* (such as the number of requests per second for the page, or the number of kilobytes of a page transmitted per second), and *scheduling policies* for each type of resources.

In order to measure the performance and the cost of each resource at different operational points, a cost function associated with each type of resources will be developed. Typically, the resource QoS is managed according to the resource model of the underlying system, which describes the resource capacity of each information server at a given moment, including CPU speed, local CPU load factor, memory, file server's capacity, network bandwidth, local area network char-

acteristics. A common abstraction to specify the capacity of an information server is in terms of bytes per second. Using this abstraction, each information server periodically determines the number of bytes per second it can deliver. Concrete formula for computing the resource capacity of an InfoFilter server is still under development and will be reported in a forthcoming paper[7].

2.6 System-Level QoS Specification

In distributed computing environments, users and applications often compete with one another for system resources. Consequently, application QoS and resource QoS may have some conflicting goals. The system perspective of the QoS model needs mechanisms to reconcile the conflicting goals between different types of applications, between heterogeneous resources, and between application perspectives and resource perspectives. For example, when a new application is started and there are not enough resources to perform it with the desired QoS, several methods can be used to free up some resources. One can degrade the QoS requirement of this new application, or degrade the QoS of a less important application that is already running. One of the main mechanisms for dealing with such resource contention problems is to define a set of end-to-end system policies. An example system policy could be to define the action that should be taken when a new application is started and there are not enough resources to perform it with the desired QoS. An obvious approach to handling competing users or competing applications is to define and evaluate the relative importance of different applications that are contending for the resource. One of the effective mechanisms is to use the price (cost) that the user is willing to pay for a server of a given quality as the measurement of relative importance.

In InfoFilter, we describe system-level QoS requirements in terms of QoS constraints. For Web applications, QoS constraints for various requests are a form of specification used to describe how a server's resources should be allocated. Typical system-level QoS specifications include specification of guarantees about byte transfer rates and page request rates, allocation of specific and relative amount of server resources to specific page requests, time-based and link-relation-based allocation of resources, scalable allocation of resources, required system throughput, to name a few. We define a QoS constraint to be a conditional QoS specification in the sense that the QoS specification must be guaranteed when the condition is true. Other issues related with system perspective of the QoS model is the multi-layer QoS enforcement architecture, which is omitted in this paper due to the space restriction.

§3 QoS Information Server

3.1 Infopipe Abstraction

The implementation of the InfoFilter QoS model described in Section 2 is a significant undertaking. To make our implementation feasible and simpler, we will build the InfoFilter QoS Information Server using the *Infopipe* abstraction, being developed in the Infosphere project, which is building the system software to support the next generation information flow applications.

Infopipe is represented by an explicit description of the syntax, semantics, and QoS requirements of the information flow. Typically the information flow is divided into logical units (potentially of variable length), and the component fields within the units are described by a microlanguage similar to C records or database schema description, which captures the information semantics. The semantics description is needed during the interpretation of the information flow. In addition to the syntax component and the semantics component, the third main component of Infopipe description consists of *the QoS requirements* of the applications producing and consuming the information flow. This is a major departure from traditional systems supporting QoS, since the QoS specification and representation are usually implicitly described or handled by the applications themselves. By attaching the QoS requirements to the information flow itself, our goal is to provide the underlying system with enough guidance to make informed decisions in resource management tradeoffs, for example, what to do under system saturation. The Infopipe software toolkit handles the translation of the Infopipe description into executable code, much the same way the Remote Procedure Call (RPC) stub generators take care of the code to marshal/unmarshal parameters into/from messages.

Each Infopipe has an input end, a processing middle, and an output end. The input and output ends are described by a *Typespec*, which is the explicit description of the syntax, semantics, and QoS requirements of the information flow at both producer end and consumer end. The processing middle transforms the information flow from the input Typespec into the output Typespec, while guaranteeing the QoS properties specified at the two ends of the infopipe. The middle also handles buffering and active push, for example. The details of Infopipe abstracts will be described in another paper [7].

3.2 Infopipe Software Toolkit

The Infopipe software toolkit contains four main components. The first component of the toolkit is *the Infopipe stub generators*, the equivalent of RPC stub generators

for Infopipes. It handles primarily the syntax and semantics for correct interpretation of the information flow. Given the Typespec of an end, the toolkit will generate the appropriate code for the parsing and generation of information flow. Our current design decision uses XML as the wire format for high-level information flows. The code to parse XML input and generate XML output is automatically generated by the toolkit.

The second part of the toolkit is *the system code* that handles QoS properties in the kernel, of the information source, the user (or another programmed robot information consumer), and the intermediate nodes along the information flow path. This could be a modification of the operating system kernel, e.g., the Quasar implementation built into the Linux kernel. Alternatively, it could be user routines written on top of the kernel, e.g., the Resource Kernel work on Windows NT. The system support typically consists of adaptive resource (e.g., CPU, memory, disk and network bandwidth) management mechanisms (e.g., control-system based Microfeedback [9]) and saturation situation management policies (e.g., tradeoffs between CPU, memory, and network bandwidth).

The third part of the toolkit is *the library code* the applications and the systems need in order to interpret the QoS requirements. In addition, it also contains the current system state, so the component nodes may exchange system status information and take appropriate adaptive action if needed. This part of the code observes the system and application behavior, and then invokes appropriate adaptation mechanisms in the operating system according to the policies specified by the application designer.

The fourth part of the toolkit is *the software supporting Infopipe composition*, forming larger or longer composite Infopipes. The main research challenge in this part of the toolkit is the preservation of predictable QoS properties during composition. Our goal is to be able to provide the application designer clear control over the QoS properties from an end-to-end perspective.

3.3 Development of QoS Information Server

There are four issues that are critical for providing QoS guarantees for fresh information delivery. The first issue is *QoS specification*, which we have discussed in Section 2. The second issue is *QoS Mapping*. The QoS specifications associated with the consumers' requests are at the application level. As the processing of a consumer's query involves resources such as CPU, memory, and network connection, the application-level QoS specifications must be mapped to resource requirements. For example, QoS parameters (such as bandwidth) have to be derived for the net-

work connection. Similarly, given a timeliness parameter, the mapping operation derives the amount of processing required so that the CPU capacity can be allocated to ensure the processing is performed at the desired rate. Figure 1 shows the QoS specification and mapping operations implemented in terms of infopipes. The third issue is *QoS enforcement*, which is mainly concerned with scheduling

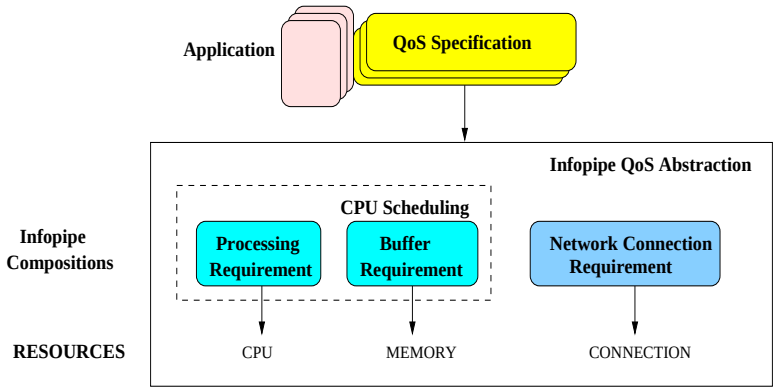

Fig. 1 Mapping application-level QoS specification to resource-level QoS requirements

shared resources during data transfer or data processing. The goal is to schedule all application threads for data transfers and data processing in such a way that they obtain their required share of the CPU and network resources in each period. This is the same as ensuring that all threads meet their deadlines. The fourth issue is *QoS adaptation*, which monitors the state of shared resources and fires the adaptive re-scheduling process whenever it is necessary. Figure 2 shows the main components used for the QoS implementation and adaptation. All concurrent applications are controlled by one adaptor for each type of system resource (most notably CPU and network bandwidth), in order to maintain fairness and stability. Each adaptor consists of monitoring task and adaptation task. The former observes the state changes and notifies the adaptation module when the amount of changes reaches certain threshold. The feedback-based configuration controller maps the adaptation decisions made by the adaptor to application-specific parameter tuning and reconfigures adaptation choices within the application. Hence, each application needs to have a corresponding feedback-based configuration controller.

More concretely, at the QoS mapping phase, the end-to-end QoS specifications of the application are translated into the third component (QoS specification) of the Infopipe Typespec. Since the Infopipe Typespec is a declarative specification of machine resources requirements, this translation effectively isolates the under-

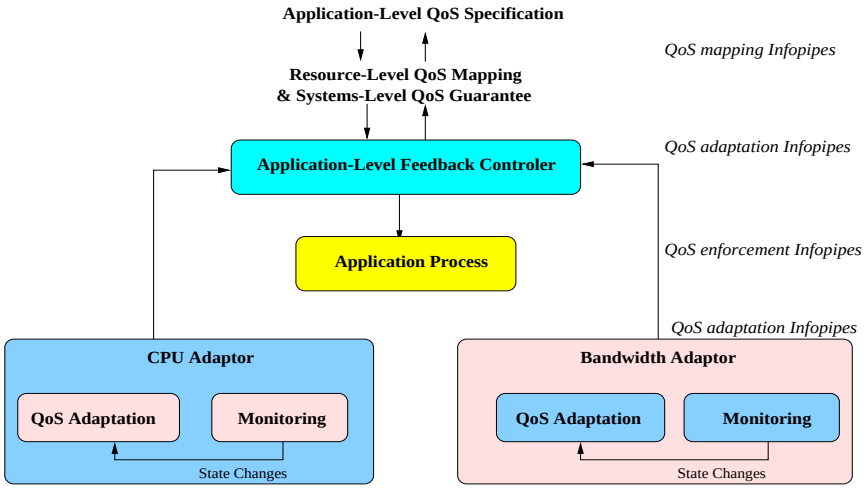

Fig. 2 QoS Implementation and Aaptation

lying hardware from the application. On the programmer side, the writing of Typespec will be handled by a GUI that speeds up the process. Then the Infopipe Typespec QoS specifications are translated into code by applying the Infosphere toolkit parts discussed in Section 3.2. In the case of QoS Information Server, we will use the part 1 of toolkit to generate the XML parser and generator code for each Infopipe. Part 2 of the toolkit consists of the system code running underlying the QoS Information Server components. Part 3 of the toolkit takes the Typespec specifications (second step) and communicates with the Part 2 kernel calls that allocate resources. In particular, the monitoring code will be generated to watch over the maintenance of QoS during execution. If saturation occurs, tradeoffs specified by the application designers will be used to invoke appropriate resource management mechanisms to recover from the saturation situation. Part 4 of the toolkit is invoked to glue together the components into the information flow grid that forms the QoS Information Server.

Consider the InfoFilter query subscription request of Example 2.1 given in Section 2.4. At the beginning, the InfoFilter query "SAR Imagery" is installed at the QoS-aware InfoFilter Server [4)] through a message. There are two parts of the query: the trigger part and the query proper. For the query proper, the InfoFilter Server is connected to the SAR Imagery information sources through Infopipes. For the trigger part, the InfoFilter Server is connected to the sensors that monitor my current location, also through Infopipes. In this example, the "2 minutes turnaround" QoS specification is translated into QoS requirements to

the SAR Infopipe. Let us assume that the SAR Images returned are about 80MB each set, and that for land movements in a car, the coverage of images results in maximum two sets of SAR data every 2 minutes. The SAR Infopipe therefore will be annotated with the QoS requirement of latency and bandwidth that will transmit 160MB within the 2-minute limit. Let us further assume that the network link to SAR is a T3 connection at 45Mbit/sec, resulting in about 40 seconds of transmission time for each set. Finally, let us assume that the InfoFilter Server has enough processors to take about 30 seconds to process each SAR data set for display, for a maximum of 60 seconds for two sets. This means that the sensor trigger must be polled at least once every 20 seconds, so there is enough time for transmitting and processing 2 sets of 80MB SAR data after movement is detected. The sensor Infopipe therefore is annotated with the QoS requirement that the sensor information must be up-to-date to within 20 seconds of the actual event being observed (location change).

With the Infopipes properly annotated, the QoS Information Server, consisting of the InfoFilter Server, the location sensors, and the SAR Imagery source, is ready to provide the QoS requested. The sensors "know" that they must notify the InfoFilter server of a location change within 20 seconds, and the sensor Infopipes take care of the system resource management to make that happen. Once notified, the InfoFilter Server fetches the SAR data sets within 40 seconds (provided by Infopipe), processes them within 60 seconds, and generates the results within the 2-minute specified turnaround time.

Note that scalability is inherent in the architecture. For example, suppose we connect to a source generating larger SAR data sets that require more processing than available within the InfoFilter Server. We upgrade the Infopipe bandwidth to the new SAR source to reduce the transmission latency, and add two high-bandwidth, low-latency Infopipes between the InfoFilter Server and a specialized processing unit for SAR data. The InfoFilter Server gets the larger data sets, sends them to the specialized processing unit, and then returns the results to the user. The configuration upgrade is transparent to the application using the QoS Information Server, and the higher performance is achieved with minimal code change at the system level. Most of the change is captured by the Infopipe QoS specifications and handled by Infosphere software.

§4 Related Work

The notion of quality of service has been studied in great detail within the context of networking [5] and multimedia systems [10]. Our work overlaps with the

research on quality of service in distributed systems where various QoS models and scheduling algorithms have been developed for supporting specific QoS guarantees. In particular, the development of our QoS model was inspired by the study on taxonomy of QoS specifications [3], the QoS support for HTTP servers and Web servers [1, 2, 6], and the Quasar QoS model [11]. However, our work differs from the previous research in a number of ways. First, our work focuses on the development of generic QoS definition and QoS specification language for distributed information flow systems. Second, we employ the Infopipe abstraction as the fundamental building blocks for implementing the QoS information flow servers. Infopipe is a natural solution for the construction of QoS Information Servers because of the close match between the information flow nature of Information Servers and the Infopipe definition, designed explicitly to support such information flows. While client/server architectures based on RPC work well for specific situations (e.g., single-company electronic commerce), Infopipes provide much more scalability and evolvability, particularly with regard to QoS support.

§5 Conclusion

We have presented the design of a distributed information flow system – InfoFilter, which implements a quality of service model. The distinct characteristics of the InfoFilter QoS model is its generic framework that unifies the QoS specifications at application-level, resource-level and system-level. Such an integrated framework enables a server to determine how consumers' requests for various web pages should be served. Several QoS enforcement mechanisms were discussed, including methods for setting priorities among various requests, association of constraints on system's resource usage.

Our future work involves the formalization of the InfoFilter QoS model, and an implementation of the InfoFilter system on top of the WebCQ system (`http://www.cc.gatech.edu/projects/disl/WebCQ`), currently operational at Georgia Tech. In addition, we plan to conduct experiments for comparing performance of the QoS-aware InfoFilter server with the WebCQ server we have built for monitoring Web information changes. Typical experiments we have in mind are measurement of behavior of QoS server with differing number of concurrent requests, comparison of throughputs and average response times of WebCQ and InfoFilter servers, as well as benefits and overhead of QoS guaranteed services.

Acknowledgment This research was partially supported by DARPA ITO under the Information Expedition program. The first three authors are also

partially supported by the Yamacraw Mission, State of Georgia.

References

1) M. Banatre, V. Issamy, F. Leleu, and B. Charpiot. Providing quality of service over the web: A newspaper-based approach. In *Proceedings of the sixth international World Wide Web conference*, 1997.

2) G. Banga and P. Druschel. Measuring the capacity of a web server. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997.

3) B. S. S. Chatterjee, M. D. J. J. Sydir, and T. F. Lawrence. Taxonomy for qos specifications. In *Proceedings of WORDS'97*, Newport Beach, California, 1997.

4) L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. *IEEE Knowledge and Data Engineering*, 1999. Special Issue on Web Technology.

5) K. Nahrestedt and J. M. Smith. The qos broker. In *IEEE Multimedia Magazine*, Spring 1995.

6) R. Pandey, J. F. Barnes, and R. Olsson. Supporting quality of service in http servers. In *Proceedings of PODC*, Puerto Vallarta, Mexico, 1998.

7) C. Pu, L. Liu, K. Schwan, and J. Walpole. The Infosphere project and the Infopipe abstraction. Technical report, College of Computing, Georgia Tech, 2000.

8) D. Rosu, K. Schwan, and S. Yalamanchili. FARA - A Framework for Adaptive Resource Allocation in Complex Real-time Systems. In *Proceedings of the 4th IEEE real-time Technology and Application Symposium (RTAS)*, Denver, USA, June 1998.

9) D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the Third Symposium on Operating System Design and Implementation (OSDI'99)*, New Orleans, February 1999.

10) A. Vogel, B. Kerherve, G. von Bochmann, and J. Gecsei. Distributed mulitmedia and qos: A survey. In *IEEE Multimedia, Vol.2 No.2*, 1995.

11) J. Walpole, C. Krasic, L. Liu, D. Maier, C. Pu, D. McNamee, and D. Steere. Quality of service semantics for multimedia database systems. In *Proceedings Data Semantics 8: Semantic Issues in Multimedia Systems IFIP TC-2 Working Conference*, Rotorua, New Zealand, January 1999.

12) R. West and K. Schwan. Experimentation with event-based methods of adaptive quality of service management. Technical report, College of Computing, Georgia Tech, 2000.

Ling Liu, Ph.D.: She is an associate professor at the College of Computing, Georgia Institute of Technology. She received her Ph.D. from Tilburg University, The Netherlands in 1993. Her research interests are in the area of large-scale data intensive systems and its applications in distributed, mobile, multimedia, and Internet computing environments. Her work has focused on systems support for creating, searching, manipulating, and monitoring streams of information in wide area networked information systems. She has published more than 70 internal journals or international conferences, and has served on more than a dozen of conference program committees in the area of data engineering, databases, and knowledge and information management.

Calton Pu, Ph.D.: He is a Professor and John P. Imlay, Jr. Chair in Software at the College of Computing, Georgia Institute of Technology. Calton received his PhD from University of Washington in 1986. He leads the Infosphere expedition project, which is building the system software to support the next generation information flow applications. Infosphere research includes adaptive operating system kernels, communications middleware, and distributed information flow applications. His past research included operating system projects such as Synthetix and Microfeedback, extended transaction projects such as Epsilon Serializability, and Internet data management. He has published more than 125 journal and conference papers, and served on more than 40 program committees.

Karsten Schwan, Ph.D.: He is a professor in the College of Computing at the Georgia Institute of Technology. He received the M.Sc. and Ph.D. degrees from Carnegie-Mellon University in Pittsburgh, Pennsylvania. He directs the IHPC project for high performance cluster computing at Georgia Tech. His current research addresses the interactive nature of modern high performance applications (i.e., online monitoring and computational steering), the development of efficient and object-based middleware, the operating system support for distributed and parallel programs, and the online configuration of applications for distributed real-time applications and for communication protocols.

Jonathan Walpole is a Professor in the Computer Science and Engineering Department at Oregon Graduate Institute of Science and Technology. He received his Ph.D. in Computer Science from Lancaster University, U.K. in 1987. His research interests are in the area of adaptive systems software and its application in distributed, mobile, multimedia computing environments. His work has focused on quality of service specification, adaptive resource management and dynamic specialization for enhanced performance, survivability and evolvability of large software systems, and he has published extensively in these areas.