

IQ-Services: Resource-Aware Middleware for Heterogeneous Applications

Zhongtang Cai, Greg Eisenhauer, Christian Poellabauer, Karsten Schwan, Matthew Wolf
{ztcai,yuanchen,eisen,chris,schwan,mwolf}@cc.gatech.edu
College of Computing
Georgia Institute of Technology

Abstract

Heterogeneous computing platforms constitute a challenging execution environment for distributed applications. This paper presents a 'systems' view of effective platform usage, by demonstrating the need for application software to be continuously 'aware' of the resources currently available on their underlying heterogeneous computing platforms. Our approach to the implementation of resource awareness is one that (1) provides a 'thin' middleware layer of resource aware services that permit applications to react to changes in resource availability and resources to be managed in accordance with application needs, and that (2) develops compiler- and application-level techniques for dynamic 'service morphing', the goal being to make it easy for application-level services to adjust to runtime changes in application needs or in platform resources. The specific results presented in this paper are focused on large-data applications, for which the IQ-Services 'morphing' layer implements the data manipulations necessary to permit wide-area interactive or multimedia applications to proceed smoothly despite variations in underlying computing and network resources. IQ-Services interact with the systems layer via dynamic performance attributes, and end-to-end implementations of such attributes also enable clients to interact with servers. This results in a cooperative approach to data management for the large-data applications targeted by our research. We present and experiment with sample IQ-Services implemented for a wide-area collaboration in which multiple end users utilize different graphical views of the data being produced by a scientific simulation. The testbed used in experiments is comprised of high end machines connected via a local area backbone, remote machines connected via the Internet, and 'home' machines connected via a DSL link. Experimental results demonstrate substantial performance improvements attained by coordinating network-level with service-level adaptations of the data being transported and by permitting end users to dynamically deploy and use application-specific services for manipulating data in ways suitable for their current needs.

1 Introduction

Research vision. Imagine an autonomous robot which upon running low on power 'sheds' all possible tasks to its collaborators so that it can continue broadcasting critical information to nearby machines, since it is the only unit able to view a safe passage out of an emergency site. Consider a tablet PC that smoothly reconfigures itself into being one of many panels in a video wall when its owner 'snaps' it into place along other such devices. Finally, imagine a real-time collaboration that continuously adjusts its large-data streams in response to currently available network resources.

While such 'morphable' devices or applications sound futuristic, the need for systems' and applications' dynamic self-modification is already well-established, and existing systems are beginning to demonstrate some of these capabilities. Cellphones with built-in bluetooth devices can save power by dynamically switching from cell modem- to bluetooth-based communications [25], as when end users synchronize their phones' daily planners with their desktop-resident calendars. The continuing merger of PDAs and cellphones is causing industry to develop new techniques for dynamically extending such devices' capabilities, trivially exemplified by the ability to download new plugins for phone-resident browsers and more importantly, exemplified by the runtime deployment of image filters to cellphones to enable real-time video conferencing in the presence of varying bandwidth availabilities[29, 31]. The large data volumes in scientific collaborations have long forced end users to employ dynamic methods for controlling the amounts of data exchanged between collaborators connected via heterogeneous wide-area networks[11, 31], and industry is

¹This work is supported in part by the NSF ANIR and ACIR programs and by the DOE MICS High Performance Networks program.

now seeking to create ‘on-demand’[10] solutions for large commercial applications using Websphere, TPF, or similar infrastructures.

Given this background, the technical problems addressed by our work are:

- the ability to run high end applications in environments that may not have sufficient resources; while
- continuously meeting application needs and environmental/resource constraints, including power budgets, and end-to-end quality of service guarantees (e.g., timing constraints); coupled with
- developing application-level functionality that can exploit these capabilities for distributed applications.

By grounding our research in challenging application domains like real-time collaboration and remote sensing, we hope that our solutions will “close the loop” by integrating system-level information about resource constraints, with middleware-level options to ‘morph’ services, with application-level opportunities for making tradeoffs and choices about how to best meet current application requirements.

Research overview. Our research complements current industry efforts, which typically focus on service interoperability (using HTML/XML or SOAP) and on ubiquitous service deployment (e.g., with Java technologies), and which oftentimes target specific platforms like PDAs or cellphones or like web-based computing infrastructures (e.g., IBM’s Websphere). Specifically, our work is developing and extending the basic compiler, systems, and middleware technologies that underly morphable – self-modifying – systems and applications. These software technologies will provide methods for dynamically modifying the software services that run on distributed computing platforms, throughout their lifetimes: when first deployed, when carrying out some newly defined task, or when subjected to runtime changes. Our motivation for creating technologies for self-modification are straightforward. First, if distributed software services are going to be truly useful, then this implies that they must deliver their functionality invisibly to end users, not requiring their explicit involvement. Second, this also implies that service delivery is subject to situation-dependent, end-to-end quality requirements and that such requirements must be met in the presence of resource constraints like like cost and power. In other words, services cannot at all times deliver all possible functions to all end users, and they cannot capture or deliver information in all forms possibly required by their potential uses. Self-modification is the way we propose to address these problems.

Our solution approach is to (1) make services continuously aware of their requirements and of their execution and platform constraints, and (2) have them self-adapt or self-modify in response to changes. While building on previous research on adaptive systems[1, 32, 24] and on dynamic program or system specialization, we will not simply extend such work. Instead, we are hoping to create entirely new ways in which software services may be adapted, which we term *service morphing*. Specifically, rather than requiring that code or applications are explicitly written to be adaptable, we are developing new combined compiler- and system-level techniques that dynamically generate and deploy code modules into distributed, embedded devices and applications. The services implemented by these code modules are *morphable* in that across two instances of time, the same service may be realized by entirely different code, often exhibiting entirely different properties in terms of its tradeoffs concerning performance and resource usage. Such code may also offer new functionality, as exemplified by a sensor service that morphs from one issuing raw sensor data to one issuing data filtered on behalf of a specific end user. For example, in a distributed scientific collaboration, an application may dynamically install entirely new data ‘filters’ into the IQ-Services layer of our large-data transport middleware. application-specific ‘filters’ installed in the IQ-Services middleware layer are driven

Challenging applications. An important attribute of our work is its concurrent exploration of realistic applications for the technologies being developed. In the embedded systems domain, we consider in detail future applications in autonomous robotics, which are characterized by (1) their high performance demands like remote sensing/vision and (2) the high levels of change they experience, due to mobility, resource changes, or changes in requirements or environmental conditions. The goal of service morphing, then, is to enable these applications to continuously operate at dynamically defined levels of Quality of Service. Furthermore, since these applications operate in pervasive environments, service morphing must be done ‘cooperatively’, that is, our solution approach must take into account the multiple devices and interacting components affected by such actions.

An application used in our research and highlighted in this paper concerns scientific collaboration. Here, end users wish to jointly inspect and manipulate scientific data when they meet, where meetings may take place in labs, in the field, next to certain instruments, or in interactive meeting rooms. Scientists use devices that range from handheld PDAs to high end displays, where at any one time, a mix of device capabilities is present and where network connectivities can vary drastically (e.g., contrast a handheld, wireless-connected device with a graphical display connected via a high end Intranet). Further, real-time collaboration implies substantial data movement and computation, since many

devices can neither display all of the rich data used in experiments nor can they run all of the computations necessary to transform the data from its raw form to the forms needed by each end user. In fact, even a simple statement like ‘look at the 2D bond force representation’ of this molecular data may involve accessing and manipulating gigabytes of data.

2 The Idea of Service Morphing

Approach: integrating middleware, compiler, and systems technologies. Service morphing targets resource limitations and runtime change in distributed, embedded applications and systems. Our approach to realizing future morphable services integrates middleware, compiler, and systems technologies. We choose a component-based framework as our underlying software architecture, thereby leveraging the fact that middleware has become the basis for most distributed and adaptive application development and deployment. Compiler-based analyses target the middleware components (and their interactions) that implement morphable services, and runtime code generation and binary rewriting are the techniques used to implement service morphing. The actual morphing techniques to be developed include dynamic component (re)deployment, (re)specialization, and (re)partitioning. Such actions will be supported by system-level mechanisms that efficiently carry the performance, usage, and requirements information needed for runtime component morphing, principally addressing components’ processing and communication actions. The intent is for self-modifying components to be able to acquire runtime information about current resource availabilities and Quality of Service demands. While developing these software technologies, we will concurrently explore new application-specific techniques and methods that take advantage of morphable software services, targeting remote sensing and autonomous robotics applications.

Toward morphable services. The problem is how to combine the power of multiple machines to jointly perform the computations and data manipulations necessary for running applications like real-time sensor processing or collaborative real-time data manipulation and display. Our technical approach is *cooperative* morphing. Viewed from a high level, two attributes characterize this approach: (1) multiple machines cooperatively execute even single application components, and (2) the software implementing the component across multiple machines is morphed during execution and/or upon deployment. More specifically, service morphing targets the ‘code modules’ used in the execution of a distributed application. This somewhat low level focus reflects our emphasis on quality of service attributes like end-to-end latency for real-time collaboration or energy efficiency for embedded applications. The techniques we are developing are challenging in their integrated use of compiler, systems, and middleware technologies:

- *middleware support*: middleware – IQ-Services – explicitly represent, describes, and executes morphable services, and it has the mechanisms needed to (re)deploy, (re)specialize, and (re)partition them at runtime; middleware also provides the basic mechanisms needed for naming, storing, and accessing service implementations, and for maintaining the meta-information needed for service morphing;
- *system support – Q-fabric*: to guarantee desired end-to-end properties, when a program component is deployed onto a certain device, this deployment must consider local resources as well as the remote cooperating machines’ resources; since the goal of service morphing is to continuously deliver certain levels of quality, we will construct system-level mechanisms that automatically ‘link’ the cooperating, distributed machines to enable joint management of their resources; specifically, the ‘Q-fabric’ underlying system support automatically establishes the kernel-level communication channels and the necessary dynamic monitoring and adaptation functions required for distributed, runtime quality management[17];
- *multiple morphing techniques: QoS-directed*: four morphing methods will be investigated: (1) dynamic (re)deployment, driven by Q-fabric-based runtime adaptation algorithms, based on changes in resource availabilities and/or application needs or environmental conditions, (2) dynamic code (re)partitioning: rather than viewing each program component as an unalterable entity, we adopt a compiler-centric view in which such a component’s code can be partitioned across cooperating machines at runtime, based on dynamically available program meta-information; an example is the dynamic partitioning of an image processing pipeline that offloads power-intensive instructions from a power-poor to a power-rich device[30, 3]; (3) code (re)specialization: to maintain certain end-to-end properties, like bounded delay, we will also specialize program code; an example is the replacement of high precision ARM with lower precision THUMB instructions in a cooperating modules of an image processing pipeline, to effect precision vs. performance tradeoffs; (4) optimization for event-intensive embedded applications (e.g., robotics), ordering and concurrency: using dynamic analyses of event dependences, we will devise optimizations for aggregating event handlers, dynamic optimizations for linearized control flow

in the absence of events, and determine and use slack for relaxing realtime deadlines to conserve power;

- *morphing for safety and protection*: a key issue in the cooperative use of multiple devices is to ensure trust in the programs dynamically deployed onto machines; we address this issue by (1) code (re)specialization, coupled with (2) system-level mechanisms that ensure address and timing safety for dynamically deployed code modules; code specialization ensures address isolation for code modules linked as libraries; protection methods ensure address isolation and timing safety for code modules placed into different protection rings, as more generally planned by initiatives like HP's Secure Platform Architecture;
- *future hardware and applications*: to increase the opportunities for runtime morphing and optimization, we will (1) develop new hardware techniques that provide morphable services with additional opportunities for changing behaviors and making runtime tradeoffs, and (2) develop novel application-level methods to integrate hardware-, with system-, with middleware-, with application-level measures of the utility of service morphing actions.

In the remainder of this paper, we focus on real-time scientific collaboration for wide-area networks. We present the IQ-Services middleware layer that facilitates runtime service morphing for such distributed applications. Results attained on heterogeneous networks demonstrate the viability and utility of our approach.

3 Real-time Scientific Collaboration on Wide-Area Networks

Distributed applications that 'stress' wide area networks include telepresence, remote collaboration and visualization, remote instrument access and control, and real-time monitoring and surveillance. Problems arise both from their large data volumes and from their interactive nature, the latter requiring data to be transferred with low latency and high predictability. Factors contributing to these problems include the heterogeneity and dynamics of underlying networks, the lack of support for QoS at the network level, and TCP's end-to-end congestion control that results in bursty network traffic coupled with the delivery of unstable QoS over time[19]. Factors also include the difficulty of measuring available network bandwidth, especially when collaborators or remote users must utilize shared substrates like the Internet.

We next describe and evaluate middleware-based methods of providing to interactive wide area applications the data they need at acceptable levels of quality. The idea of the IQ-Services is to have middleware execute application-defined, morphable, services for data filtering, scheduling, and transformation, and to continuously adjust service behavior in accordance with current platform conditions. Platform behavior, focused on network conditions, is captured with dynamic bandwidth measurement[15] techniques. Morphable services are installed at runtime and under application control, initiated by applications and on the machines used by middleware. An example is a data filtering service installed by a client on a machine acting as its data source, thereby giving the client complete control of the data sent to it. A concrete instance is a data downsampling filter used by a visualization client, where the filter continuously varies data resolution in order to maintain acceptable data transmission rates, despite variations in available network bandwidth[11, 9]. Other examples include data reordering, prioritization, and elimination[27, 9], the use of application-specific compression methods[28], and data transformations that implement tradeoffs in the amounts of processing vs. data volumes in the overlay networks middleware uses for service execution[3].

We next briefly describe the service model used at the middleware level, the architecture of the IQ-Echo network-aware middleware providing this service model, and a service-aware communication protocol underlying such middleware. This protocol, termed IQ-RUDP[9], provides instrumentation less general than but akin to the instrumented Linux protocol stacks developed in the Web100 project. In contrast to such work, however, IQ-RUDP uses dynamic performance attributes associated with communications to convey monitoring and control information across the middleware/protocol boundary. Per message monitoring information includes current RTT (round trip time) and loss rate. Control information includes desired loss tolerance and packet markups with priorities.

The ability to 'drive' dynamic changes in middleware services with protocol-level information is demonstrated with an interactive high performance application, termed SmartPointer[28]. This application permits remote users to collaboratively view and manipulate data produced by molecular dynamics simulations[28]. Users can dynamically specialize data by using middleware-level transformation and filtering services. These services are dynamically adjusted in accordance with network-level changes in behavior. Experimental evaluations of this approach are performed across wide-area network links and on the NetLab network emulation facility[13].

This paper's results leverage earlier work by our group[9] in which we demonstrated that (1) coordinated application/middleware- and network-level adaptations can outperform application- or network-only adaptation methods, using performance metrics like jitter, for instance, and that (2) it is advantageous to permit the network level to 'drive' (i.e., initiate) such adaptations due to its ability to adapt at higher rates and therefore, better accuracy than purely

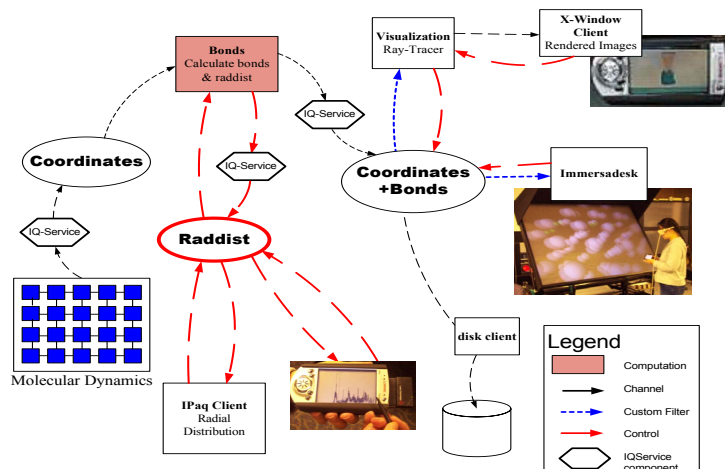


Figure 1: SmartPointer System Overview

application-initiated adaptations[9]. Earlier work, attained via emulation and simulation, also demonstrates the ability of performance attributes to coordinate network- with middleware-level adaptations. Finally, in [18], we describe Linux-based, kernel-level mechanisms that enable the asynchronous and efficient exchange of performance attributes across the protocol, kernel, and middleware boundaries.

The remainder of this paper is organized as follows. In Section 3, a sample distributed application using IQ-Services is outlined, including a description of its performance requirements and of the IQ-Services associated with its use. In Section 4, the architecture of IQ-Services is described. Experimental results attained on wide area networked machines and on the NetLab emulation facility appear in Section 5. Conclusions and future research are outlined in Section 6.

4 Wide-area Scientific Collaboration

There are multiple, ongoing wide-area collaborations in high performance computing, ranging from high profile efforts like Ligo[14] and DOE's Terascale Supernova Initiative (TSI)[5] to adhoc collaborations between specific sets of researchers. Collaboration tools like Access Grid and grid portal efforts, and communication infrastructures like ECho[6] are evidence of the importance of remote collaboration to the broader CS and scientific communities. The distributed collaboration code used in this paper is representative of such applications in several ways:

- The underlying computing/network infrastructure is heterogeneous, linking high end machines with workstations and even portable devices, with networks ranging from high speed LANs to wide area to wireless connections.
- The amounts of data produced, transported, and consumed are large, expected to be up to 100gb/sec in applications like TSI, but in our current work, we use more moderate data speeds.
- For real-time collaboration, fresh or new information may be more important than complete detail, so we share real-time requirements with applications like multimedia and video services.

Figure 1 shows a proto-typical, distributed collaborative visualization applications. It represents a many-to-many data-flow, with data originating in a parallel simulation, passing through multiple dynamically-instantiated filters, and ultimately delivering to a variety of clients. In order to make the collaborative environment function efficiently, the delivery of data to the various clients must be coordinated; if the information for one collaborator consistently lags that of the others, the collaboration activities will be compromised.

Control loops for the data filters further complicate the system, especially as there is the capability for one collaborator to drive the annotation of the data for the others. Soft real-time constraints may drive the collaborators to choose consistent frame rates (to insure data freshness) over the actual resolution of the data. Several of the filters described in this work implement exactly these sorts of trade-offs, and go a step further to document the need to coordinate application-level adaptation with transport adaptation.

In summary, performance criteria for the class of distributed collaboration applications we have chosen to test are: information freshness[20], end-to-end information quality, and fairness, particularly across the multiple connections maintained between even a single source-sink pair of machines in collaborative applications.

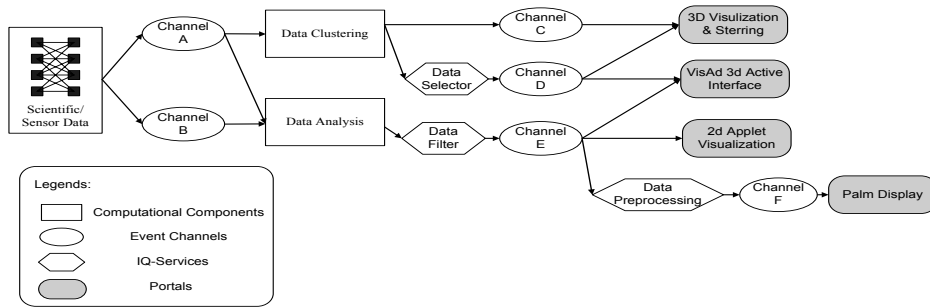


Figure 2: Typical Application Scenario

5 IQ-Services: Middleware Architecture for Morphable Services

For large-data applications like scientific visualization and collaboration, IQ-Services are application-specific services that are associated with data transport middleware. Figure 2 shows a typical scenario in which application-level data is distributed to remote collaborators and is manipulated by computational components like data clustering or data analysis[16]. The role of IQ-Services in this scenario is illustrated by the additional data filters and selectors associated as handlers with the event channels used for data distribution and transport. As evident from this figure, IQ-Services do not replace application-level components like data analysis or cluster algorithms. Instead, they form a ‘thin’, efficient layer of application-provided functionality that is placed ‘into’ middleware by applications, the purpose of which is to allow middleware to manipulate data on its path from providers to consumers and to do so in conjunction with information about network behavior provided by communication protocols. In IQ-Echo, this layer is comprised of dynamically created and deployed ‘event handlers’.

Performance attributes implement IQ-Service/protocol interactions, as well as end-to-end performance-relevant interactions between information providers and consumers. Examples of such interactions include a client’s use of performance attributes to set parameters in a server’s data filter, and a server-side instruction of the middleware handler to upsample the data sent since additional network bandwidth is now available. End-to-end and cross-layer interactions via performance attributes are depicted by the solid, vertical arrow in Figure 3, which also shows the multiple levels of abstractions used in IQ-Echo to map an application-level event submitted to an event channel to a communication protocol stack and socket used by a specific information source-sink pair: after event submission, the event is mapped to a lower-level facility called ‘communication manager’, which then sends the event as a message to one of multiple communication protocols used in event transmission. The instrumented IQ-RUDP protocol used in the measurements presented in this paper uses performance attributes (1) to provide feedback to the application-level handler also shown in the figure and (2) to adjust its data transmission behavior (e.g., using attributes to not retransmit certain data upon detected loss). In addition, network measurement capabilities may be directly associated with communication services, depicted in Figure 3 by the ‘measurement’ methods layered between event management and communication manager. By placing such measurement methods ‘into’ the communication stream, measurements can be performed using the application’s native communications, rather than additional packet trains generated for such purposes. This is particularly suitable for the streaming data applications targeted by our work, but may need to be modified to also enable active bandwidth measurements for bursty application-level communications.

IQ-Services adjusted via performance attributes may reside on clients, servers, or an intermediate nodes, thereby

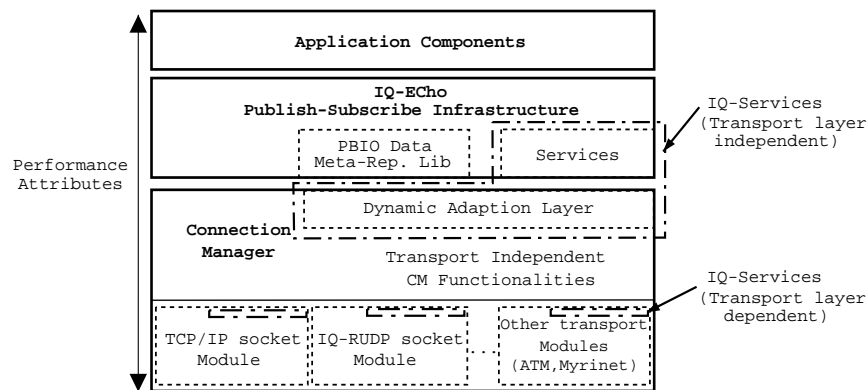


Figure 3: System Architecture Overview

forming an overlay network. Typical configurations of overlays used in interactive applications are described in [16, 2]. We are not concerned with how to best map overlays to communication/processor networks, but note that an aspect of overlays inherent to our work is that their dynamic nature implies that it must be possible to create and then use services dynamically, when desired by applications or when indicated by substantial changes in network or machine resources. IQ-Echo supports this by permitting the dynamic definition and installation of services, using dynamic binary code generation techniques[6].

We will not describe the full set of layers depicted in Figure 3. Instead, we next briefly outline the IQ-RUDP transport layer that implements coordinated network and application-layer adaptation. The experiments with this simple layer demonstrate the utility of this approach in maintaining application performance in the face of changing network conditions.

5.1 IQ-RUDP

IQ-RUDP is an open transport protocol that is intended to facilitate end system and application-layer adaptations to network conditions. To attain TCP-friendly behavior, it also runs its own congestion control algorithm. The distinctive features of IQ-RUDP can be summarized as follows:

- Exposing performance metrics. IQ-RUDP exposes certain transport-layer performance metrics to the higher layer by means of IQ-Echo attributes. Metrics such as bandwidth, round-trip time (RTT) and loss ratio can be exploited by the IQ-Services when adapting the data being transported.
- Callbacks. Applications can register some callbacks to be called by IQ-RUDP, along with the conditions under which they should be triggered.
- Adaptive and application-controlled reliability. One of the adaptations often adopted by applications is to lower reliability requirement. IQ-RUDP provides prioritized reliability control for each application-level message.
- Coordinated adaptations between the application and the transport. IQ-RUDP first proposed the idea of coordination between application adaptation and the transport behavior[9]. The advantages of coordinated adaptation have been demonstrated in cases where the application has limited adaptation granularity, or where application interests conflict with network friendliness.

The primitives provided by IQ-RUDP compose the basis for higher-layer control and adaptations. However, the indirection involved in the translation between application-level information and transport-level metrics either limits or complicates the direct use of IQ-RUDP primitives by applications with rich application semantics. Moderating between lower-level adaptation primitives as provided by IQ-RUDP and application-level semantics and desires is largely the role of the Dynamic Adaptation Layer (DAL), embedded into our current implementation of IQ-Services.

The adaptation types provided by IQ-Services and their methods we demonstrate can be generalized and implemented based on other transport protocols, including the widely used TCP. For instance, TCP socket buffer auto-sizing[26] can be utilized to improve or control transport level throughput. Other autotuning protocols, e.g. many modified TCP protocol[23, 4], can be easily fit into the IQ-Services architecture. Finally, performance metrics, including RTT, available bandwidth, and throughput etc., can be exposed by DAL without support from IQ-RUDP. Specifically, available bandwidth can be measured using standalone services like Pathload.

5.2 Dynamic Adaptation Layer

The Dynamic Adaptation Layer (DAL) implements the monitoring and adaptation methods used in interactions between IQ-Services and the transport layer. Adaptation examples include packet reliability control and coordination across multiple connections, all of which are supported by the quality attributes and adaptation methods realized by DAL.

Adaptation methods.

- Packet Reliability Control
Applications can define different levels of packet reliability requirement and packets with different reliability level will be handled accordingly. In our experiments, the data packets with highest reliability are sent out in a timely and reliable manner, while data packets with lowest reliability are sent only when there are sufficient resources.
- Congestion Avoidance
When supported by the transport layer, as in the new IQ-RUDP and in many TCP variances, DAL would control the transport layer's sending rate and congestion control behavior, so as to avoid possible congestion, recover faster from existing congestion, and improve throughput while still maintaining TCP-like fairness.
- Coordinating Multiple Connections

Middleware support for coordination between multiple channels for collaborator-friendly communication can be vital in the presence of resource contention. Consider a scenario in which several peers are communicating with a cluster of servers sharing a common bottleneck (e.g. the MD server cluster in the SmartPointer application). Because of heterogeneity, different connections to the server cluster will have different RTTs, and therefore, they will not be treated equally by the network layer. Specifically, clients with larger RTTs to the server would receive smaller fractions of the bottleneck bandwidth, especially when contention is heavy.

Coordination between multiple channels is performed in the DAL, not in the transport or network layers, since a connection in these two layers (E2E link for transport layer, hop by hop link for network link) has no knowledge about other connections. Coordination heuristics are provided by upper layers to adjust the transport layer's behavior in an application-friendly fashion.

6 Evaluation

This section presents experimental results that demonstrate the benefits of application- and network-level coordination in managing network resource availability. These results include:

- demonstrations that coordinated network- and middleware-level adaptation can substantially improve the performance of wide area applications; and
- realistic examples of adaptations performed for interactive high performance applications, focusing on online collaboration via large data sets.

6.1 Testbed

The testbed used to evaluate coordinated network-and middleware-level adaptation consists multiple wide area network links and local area network links. The two local (.cc.gatech.edu) nodes **smartin** and **guadeloupe** are connected to the backbone with 100Mbps and 1Gbps links, respectively. In another building (GCATT), **vega** and **altair** both have 1Gbps links up to the backbone. The Round-Trip Times (RTTs) between smartin and guadeloupe are normally in the range of 0.06ms to 0.13ms. The RTT between GCATT nodes and CERCS nodes, is slightly larger (0.2ms to 0.4ms). A node at Louisiana State University is connected with Georgia Tech with a 10Mbps link.

6.2 Experimental Results

Coordinated packet reliability control. This experiment shows the effectiveness of coordinating reliability control between IQ-RUDP and IQ-services, for collaborative applications that use application-specific methods of dealing with data loss. In this example, a bondserver (as described above in Section 4) application component is sending data over a lossy link (*i.e.*, a connection from a home DSL machine to a campus machine, **guadaloupe**). There are three types of application messages, and in the order of importance, they are: messages with atoms and bonds information [AB], messages with atoms information only [A], and messages with bonds information only [B]. In addition, different types of messages are grouped together according to the pattern (AB, B, B, A, B). There is often redundant information among the messages in one group. Overall, AB messages have to be delivered, both type A and B messages can be lost, where type B messages have lower priorities than those of type A.

In the coordinated reliability control algorithm, IQ-RUDP defines three levels of packet reliability requirement and handles them in the following way:

- it discards the lowest priority messages without transmitting them;
- it attempts to transmit the second priority messages but will not try to retransmit them if they are lost; and
- it guarantees the reliable delivery of the highest priority messages.

In this set of experiments, we set a target rate for the type AB message in each experiment. When there are significant packet losses and the target rate isn't achieved, the application adaptation sets certain portions of type B messages to the lowest priority, and sets the type A messages to the second priority.

Results in Figure 4 show that the algorithm successfully achieves the different target rates set a priori, at the cost of an increased loss ratio of type B messages. Type A messages, which are also much smaller than the other two, have very few losses.

Coordinating the transfer of two IQ-RUDP connections. In this experiment, we coordinate, at the DAL layer, the transfer of two IQ-RUDP connections used by a single application. A typical example in remote collaboration are the co-existence of control and data connections, or the co-existence of large-data connections (e.g., visualization data) with video connections (e.g., for video conferencing). This experiment demonstrates that bandwidth measurement at the IQ-RUDP layer can be used to better synchronize such connections.

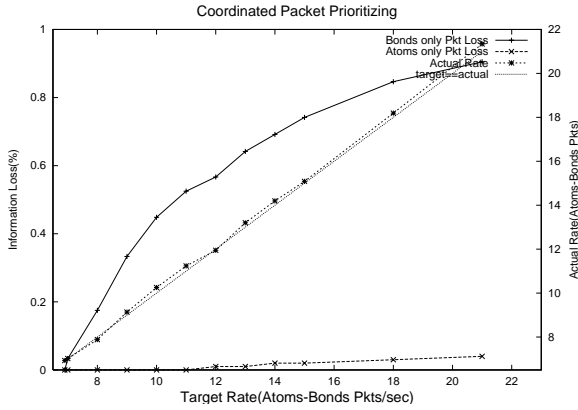


Figure 4: Prioritized Reliability Control

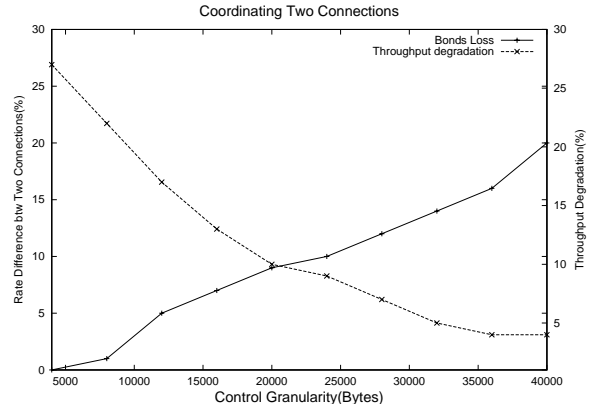


Figure 5: Coordination between Connections

The application has two threads sending data to two different receivers, and the receivers are intended to be synchronized in terms of the data they see. The two connections share a bottleneck link and have different RTTs and therefore have different throughput.

When the application starts, it establishes each IQ-RUDP connection and measures the available bandwidth. When the throughput information of both connections is available to the application, the application slows down the faster connection in proportion to the ratio of their throughput measurements.

In Figure 5, the X-value is the size of the data unit on which the sending rate is controlled. The left Y-value is the receiving rate difference between the two receivers. The right Y-value is the throughput degradation of on the slow connection. We can see that finer grained control achieves better synchronization, at the cost of slowing down the slower connection, i.e., not fully utilizing the network bandwidth.

In separate experiment, we use the DAL layer to make two connections share a bottleneck link fairly, using the bandwidth and RTT measurements at the connection layer. The application adjusts the sending rate based on the RTTs of the two connections. The adjustment algorithm is based on the TCP throughput formula, which shows that two connections sharing the same bottleneck link have throughputs that are inversely proportional to RTT^2 . The results we get are very similar to those on Figure 5, although a different IQ-RUDP layer measurement is used.

Packet pacing. This experiment shows how the ‘pacing adaptation’ based on measured network metrics can both improve and smooth throughput. When the bond server transfers large amounts of data to some specific client, we inject cross-traffic into the network, thereby decreasing available network bandwidth. With IQ-Services, network

| | Message Delivery Rate(f/s) | Normalized Standard Deviation | Jitter(ms) |
|--------------------|----------------------------|-------------------------------|------------|
| Without Adaptation | 443.99 | 0.27 | 2.41 |
| With Adaptation | 599.39 | 0.17 | 1.76 |

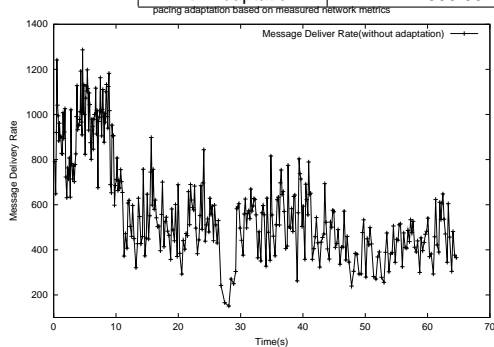


Figure 6: Without Packet Pacing Adaptation

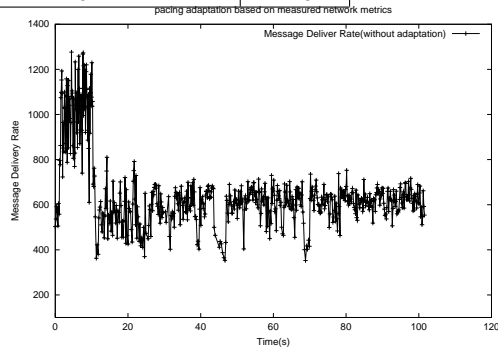


Figure 7: With Packet Pacing Adaptation

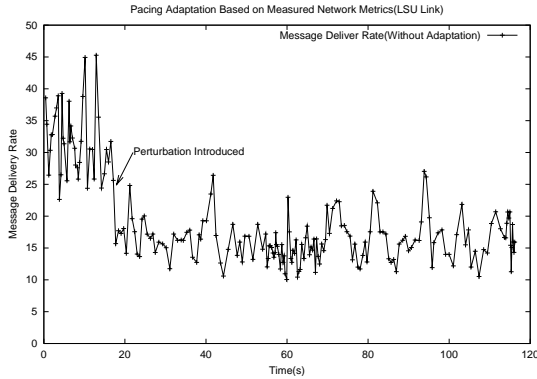


Figure 8: Without Packet Pacing Adaptation (LSU Link)

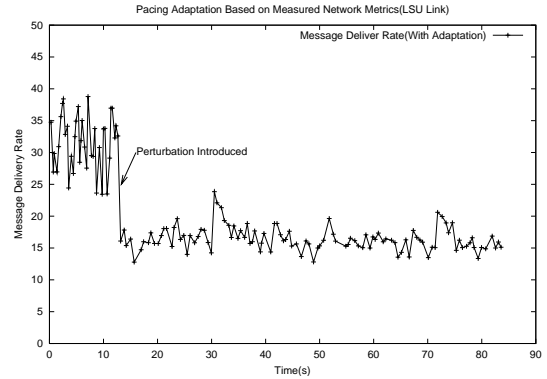


Figure 9: With Packet Pacing Adaptation (LSU Link)

Table 1: Packets Pacing Adaptation - Performance Comparison(LSU Link)

| | | Message Delivery Rate(f/s) | Normalized Standard Deviation | Jitter(ms) |
|----------------------------|--------------------|----------------------------|-------------------------------|------------|
| Exp1(Figure 8, Figure 9) | Without Adaptation | 16.11 | 0.20 | 61.9 |
| | With Adaptation | 16.45 | 0.17 | 55.6 |
| Average Over 5 Experiments | Without Adaptation | 16.05 | 0.18 | 60.2 |
| | With Adaptation | 16.25 | 0.17 | 54.9 |

metrics are exposed through performance attributes to middleware and to application modules, thereby making both ‘aware’ of changes in network status and permitting them to adjust their behavior accordingly. In this experiment, they adjust the way in which packets are sent out (packet pacing). The packet size used is 66KB, and the injected cross-traffic over the 1Gbps link is 650Mbps, generated by Iperf.

Figure 6 illustrates the measured message delivery rate at the client side from a server that is unaware of the cross-traffic and can not adapt to changing network status. Figure 7 shows much better performance after the cross-traffic is injected, since in this case, the server is made aware of the measured loss rate and the available bandwidth via IQ-Services and then paces its packets accordingly (at a better rate). The idea is to ensure that its throughput does not exceed the available bandwidth, thereby resulting in reduced congestion and packet losses. The message delivery rate improves from 434f/s to 599f/s, and it is much smoother: the normalized standard deviation of the message delivery rate decreases from 0.27 to 0.17, and jitter decreases from 2.41ms to 1.76ms.

These results are attained using bandwidth measurement techniques described in [21]. These techniques are used to predict the available bandwidth in next time slot. In comparison, loss rate reflects current network status. The idea is for the server to adapt quickly to network changes, but to avoid oscillations in such adaptations by use of bandwidth prediction.

A similar experiment is performed between **smartin** and **resource.rri.lsu.edu**, as shown in Figure 8 and Figure 9. This link has more fluctuation in terms of available bandwidth and RTT, since there is other significant and often unpredictable traffic that also impacts the link besides the traffic we generate. Figure 8 and Figure 9 demonstrate how both throughput and jitter are improved in a typical run of the experiment. Table 1 shows the average performance improvement over 5 experiments.

Adaptive downsampling in congested networks. While packet pacing can effectively reduce congestion and maintain better message delivery rates, its benefits are limited when the available bandwidth is already below the threshold at which it becomes impossible to deliver messages as fast as required. Experimental results shown in Figure 10 and Figure 11 demonstrate the necessity and effectiveness of IQ-Services-level adaptations through data downsampling. Stated more explicitly, the next results show how runtime service morphing can substantially improve the dynamic behavior of this SmartPointer distributed application.

In this experiment, larger amounts of cross traffic (850Mbps) are injected into the network. Using the algorithm described in previous experiment, the server can only deliver 23.3 f/s to the client (Figure 10). However, since the client can characterize the specific subset of data it most requires, when congestion occurs, it can install an IQ-Service

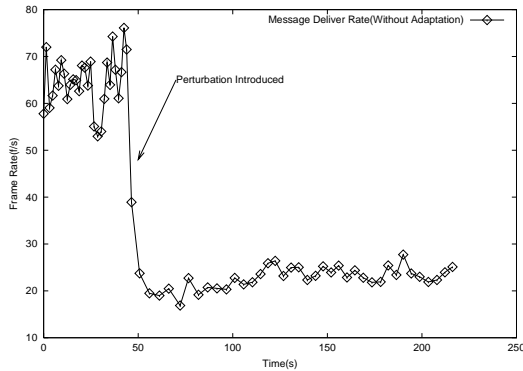


Figure 10: Without ECho Meta-data Based Filtering

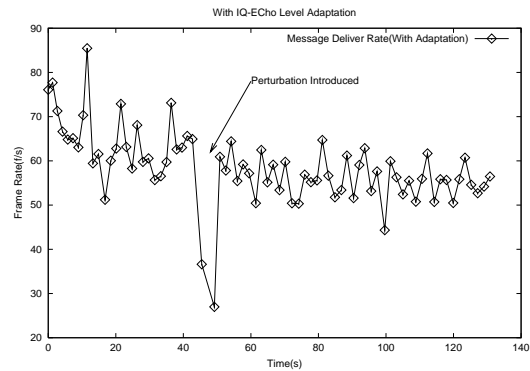


Figure 11: With ECho Meta-data Based Filtering

data filter at the server side. In this way, client can get essential data at a satisfying speed. This data downsampler is also used in the application described above in Section 4. It removes data relating to visual objects that are not in the user’s immediate field of view, resulting in much more effective data transfer. (Another useful filter is one that does data precision downsampling). In addition, this filter interacts with the network layer to adjust the data sent to available network bandwidth. In this fashion, the client receives the data it needs to continue operating with a satisfactory rate, despite changes in available network bandwidth. In this experiment, the client transfers the current position and view point of the user to the filter, which computes at the server side what data set the user is watching and transfers exactly these data sets. Figure 11 shows that the effective frame rate received is much higher than that in Figure 10 (which does not use such a downsampling filter).

7 Conclusions and Future Work

IQ-Services are an example of the morphable, resource-aware services being developed in our ongoing research. The software architecture of IQ-Services shown in Figure 3 is specific to the IQ-Echo publish/subscribe middleware used in our work, but the principle of inserting application-specific, morphable services into middleware is easily generalized and applied to both the embedded systems domain and the middleware infrastructures now under development[7]. Concerning grid middleware web-based service infrastructures, for instance, earlier work developed application-specific policy objects[12] or subcontracts[8] to control data distribution and management. Subcontracts are also part of the implementation of RMI in Java, and many modern object-based systems support the notion of meta-interfaces that may be used to alter the implementation of objects without changing their basic interfaces. In fact, in ongoing work, we are associating policy objects and data handlers with SOAP-based communications, in order to dynamically adjust data transmissions to changes in network resources or client needs[22]. In addition, while the results shown in this paper use the IQ-RUDP protocol, we expect to attain similar results for dynamic adaptations that simply adjust application-level services, without also affecting protocol-level behavior, we could use instrumented communication protocols like those provided by the web100 group[15], or with the Q-fabric system-level support, it is possible to implement ‘richer’ application-system interactions in order to better manage service morphing to meet application needs with varying runtime resources.

Our future work will utilize overlay networks to combine the lightweight data filtering and downsampling methods used in this paper with heavier-weight methods for data transformation and summarization executed by additional machines interposed into the path between data providers and consumers[16]. Such work will consider the runtime deployment of lightweight IQ-Services to dynamically utilize alternative network and machine paths from data providers to consumers. Future and ongoing work is also considering entirely different computing platforms, including embedded systems infrastructures. For such platforms, service morphing may consider entirely different quality of service metrics like energy usage[17]. In addition, our prior work has already shown that for such infrastructures, it becomes increasingly important to perform efficient dynamic code generation for runtime service specialization and optimization as part of the service morphing process[3, 30].

References

- [1] T. Abdelzaher, M. Bjorklund, S. Dawson, W.-C. Feng, F. Jahanian, S. Johnson, P. Marron, A. Mehra, and T. M. et al. AR-MADA Middleware and Communication Services. In *RTS*, 1997.
- [2] P. Chandra, A. Fisher, C. Kosak, and P. Steenkiste. Network support for application-oriented quality of service. In *Proceedings of the Sixth IEEE/IFIP International Workshop on Quality of Service*, May 1998.
- [3] Y. Chen, K. Schwan, and D. Zhou. Opportunistic channels: Mobility-aware event delivery. In *Proceedings of the ACM/USENIX International Middleware Conference*, 2003.
- [4] a. "D.Katabi, M.Handley. Congestion control for high bandwidth-delay product networks. In *ACMSIG-COMM*, Aug 2002.
- [5] DOE-TSI. Terascale supernova initiative. <http://www.phy.ornl.gov/tsi>.
- [6] G. Eisenhauer. The ECho Event Delivery System. Technical Report GIT-CC-99-08, Georgia Tech, Aug 1999.
- [7] Globus. <http://www.globus.org/ogsa/>. An Open Grid Services Architecture, 2003.
- [8] G. Hamilton, M. Powell, and J. Mitchell. Subcontract: A flexible base for distributed programming. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 69–79, 1993.
- [9] Q. He and K. Schwan. IQ-RUDP: Coordinating Application Aaptation with Network Transport. In *High Performance Distributed Computing*, July 2002.
- [10] IBM. Websphere. <http://www.ibm.com/autonomic/websphere.shtml>.
- [11] C. Isert and K. Schwan. ACDS: Adapting computational data streams for high performance. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, May 2000.
- [12] C. E. Kilpatrick and K. Schwan. ChaosMON – application-specific monitoring and display of performance information for parallel and distributed systems. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 57–67, Santa Cruz, California, May 1991. ACM Press.
- [13] J. Lepreau. The Utah Network Testbed. <http://www.emulab.net/>. University of Utah.
- [14] LSC. <http://www.ligo.org/>. LIGO Scientific Collaboration., 2003.
- [15] M. Mathis. Web100 and the End-to-End problem. <http://www.web100.org/docs/jtech/>.
- [16] B. Plale, G. Eisenhauer, K. Schwan, J. Heiner, V. Martin, and J. Vetter. From interactive applications to distributed laboratories. *IEEE Concurrency*, 6(3), 1998.
- [17] C. Poellabauer, K. Schwan, S. Agarwala, A. Gavrilovska, G. Eisenhauer, S. Pande, C. Pu, and M. Wolf. Service morphing: Integrated system- and application-level service adaptation in autonomic systems. In *Proceedings of the 5th Annual International Workshop on Active Middleware Services (AMS 2003)*, June 2003. Seattle, Washington.
- [18] C. Poellabauer, K. Schwan, and R. West. Lightweight Kernel/User Communication for Real-Time and Multimedia Applications. In *NOSSDAV*, Jun 2001.
- [19] P.Tinnakornsrisuphap, W. Feng, and I. Philp. On the Burstiness of the TCP Congestion-Control Mechanism in a Distributed Computing System. In *ICDCS*, 2000.
- [20] C. Pu. The infosphere project. <http://www.cc.gatech.edu/projects/infosphere>.
- [21] N. S. Rao, Y.-C. Bang, S. Radhakrisnan, Q. Wu, S. S. Iyengar, and H. Choo. NetLets: measurement-based routing daemons for low end-to-end delay over networks. *Computer Communications*, to be published.
- [22] B. Seshasayee, K. Schwan, and P. Widener. SOAP-binQ: High-performance SOAP with continuous quality management. In *Proceedings of International Conference on Distributed Computing Systems*, 2004. To Appear.
- [23] S.Floyd. High speed tcp for large congestion windows. Internet Draft: draft-floyd-tcp-highspeed-01.txt.
- [24] L. Sha, X. Liu, and T. Abdelzaher. Queuing model based network server performance control. In *Proceedings of the Real-Time Systems Symposium*, December 2002.
- [25] A. Snoeren, D. Anderson, and H. Balakrishnan. Fine-grained failover using connection migration. In *Proceedings of the Third Annual USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2001.
- [26] B. "T.Dunigan, M.Mathis. A tcp tuning daemon. In *SuperComputing:High-Performance Networking and Computing*, Nov. 2002.
- [27] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proceedings of the Real-Time Systems Symposium*, 2000.
- [28] M. Wolf, Z. Cai, W. Huang, and K. Schwan. Smart Pointers: Personalized Scientific Data Portals in Your Hand. In *Proc. of Supercomputing 2002*, Nov. 2002.
- [29] D. Zhou. Private communication. NTT DoCoMo Corporation.
- [30] D. Zhou and K. Schwan. Eager handlers - communication optimization in java-based distributed applications with fine-grained code migration. Proceedings of the 3rd International Workshop on Java(tm) for Parallel and Distributed Computing, April 2001.
- [31] D. Zhou, K. Schwan, G. Eisenhauer, and Y. Chen. Supporting distributed high performance application with java event channels. In *Proceedings of the 2001 International Parallel and Distributed Proceeding Symposium (IPDPS)*, April 2001.
- [32] J. Zinky, D. E. Bakken, and R. Schantz. Architecture Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, January 1997. Also see <http://www.dist-systems.bbn.com/tech/QuO>.