# Service Morphing: Integrated System- and Application-Level Service Adaptation in Autonomic Systems

Christian Poellabauer, Karsten Schwan,
Sandip Agarwala, Ada Gavrilovska, Greg Eisenhauer
Santosh Pande, Calton Pu, Matthew Wolf
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{chris, schwan, sandip, ada, eisen, santosh, calton, mwolf}@cc.gatech.edu

## Abstract

*Service morphing is a set of techniques used to continuously meet an application's Quality of Service (QoS) needs, in the presence of run-time variations in service locations, platform capabilities, or end-user needs. These techniques provide high levels of flexibility in how, when, and where necessary processing and communication actions are performed. Lightweight middleware supports flexibility by permitting end-users to subscribe to information channels of interest to them whenever they desire, and then apply exactly the processing to such information they require. New compiler and binary code generation techniques dynamically generate, deploy, and specialize code in order to match current user needs to available platform resources. Finally, to deal with run-time changes in resource availability, kernel-level resource management mechanisms are associated with user-level middleware. Such associations range from loosely coupled, where kernel-level resource management monitors and occasionally responds to user-level events, to tightly coupled, where kernel-level mechanisms import, export, and use performance and control attributes in conjunction with each resource-relevant user-level event.*

## 1 Introduction

**Background.** Imagine 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. Or consider a server system that dynamically reconfigures its support for clients when such clients' runtime behaviors or resources change, as exemplified by low-end, remote PC- or PDA-based clients that require servers to help produce high quality images for their graphical displays.

While such self-modifying systems sound futuristic, the need for systems' dynamic self-modification is already well-established, and existing systems already demonstrate some of these capabilities. In the embedded domain, some cellphones can save power by dynamically switching from cell modem- to Bluetooth-based communications [26]. The continuing merger of PDAs and cellphones causes the development of new techniques for the dynamic extension of such devices' capabilities, exemplified by the ability to download plugins for phone-resident browsers and more importantly, by the runtime deployment of image filters to cellphones to enable real-time video conferencing in the presence of varying bandwidth availabilities [33]. Remote inspection of mobile units (e.g., trucks) is now giving way to their remote maintenance, using on-board communication and computer equipment [16]. Finally, server systems or the overlay networks that amplify individual servers are becoming increasingly 'conscious' of client needs, currently addressing specific tasks like remote graphics and visualization [9, 31], but with future work already considering other assistance, such as the functionality currently provided by application servers that perform XML or HTTP processing for companies' large-scale operational information systems [15].

**Service morphing.** The key problem addressed by our work is how to continuously meet application and end-user needs, including power budgets, end-to-end QoS guarantees, and security constraints. Furthermore, to understand how application-level functionality can exploit these capabilities, we are investigating scientific collaboration, remote sensing, and operational information systems [15]. We are building on our own and others' previous work on adaptive systems [24, 25, 34] and on dynamic program or system specialization [14], but our new research differs in that we are also trying to develop techniques for the automatic creation of adaptive software services and of adaptation techniques for such services, which we term *service morphing*. Specifically, rather than requiring applications to be explicitly made adaptable, we are developing compiler- and system-level techniques that dynamically generate and deploy code modules into distributed systems 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 codes, typically also exhibiting entirely different properties concerning its tradeoffs in performance versus resource usage. Such modules may also offer new functionality, a simple example being a server that begins to encrypt the data it delivers to a certain end-user when intrusions are detected for that client, a more complex example being the dynamic remapping of an image processing pipeline from a client to a server system or to an overlay network when the clients' battery power runs low. Our goal is to provide methods for dynamically modifying the software services that run on distributed systems, throughout their lifetimes: when first deployed, when carrying out some newly defined task, or when subjected to runtime changes. The assumption is that services cannot at all times deliver all possible functions to all end users, in part because such functions may not be known until runtime, and similarly, that services cannot capture or deliver information in all forms possibly required by their potential uses.

## 2 System Description

The following sections describe the architecture of our approach. Service morphing is based on the following principles:

- services *self-modify*, where this entails the combined use of middleware, compiler, and system technologies to generate appropriate outcomes;
- self-modification may entail coordination across multiple levels of abstraction, such as the middleware and network levels [5, 7];
- self-modification is assisted by underlying platforms, including extension to existing operating system kernels [17, 19].

Service morphing is supported by three interacting components:

1. *InfoFabric middleware platform.* Flexibility in communication and processing is provided by a lightweight publish/subscribe middleware: end-users subscribe to information channels of interest to them whenever they desire, and they apply exactly the processing they need to such information, using dynamically generated 'handler' functions. The goal is to operate with levels of flexibility similar to those of Java environments, but with the performance levels needed for the high end server systems used in applications like operational information systems [15].
2. *Configurable resource management.* Changes in resource availability are dealt with by a lightweight resource management framework, called *Q-Fabric*. Q-Fabric efficiently carries the performance, usage, and requirements information needed for run-time adaptation of processing and communication actions. Jointly with middleware it also implements low cost representations of such information, termed 'attributes'. The intent is for middleware to potentially have detailed knowledge of the ways in which information is transported and manipulated before delivering it to end-users, and for the underlying Q-Fabric to coordinate system- with middleware-level actions in such endeavors.
3. *Dynamic binary code generation.* To attain high performance and to meet embedded systems requirements, new compiler and code generation techniques dynamically generate and install new handler code on the InfoFabric's and Q-Fabric's platforms. Further, such run-time code generation also specializes handler code in order to match current user needs to available platform resources.

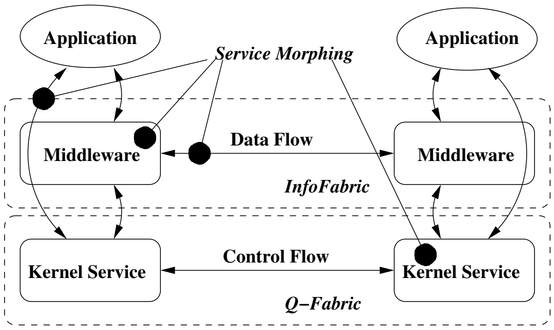


**Figure 1. System overview.**

Figure 1 illustrates the components of the service morphing approach. The InfoFabric middleware 'binds' applications with information channels of interest to them. The Q-Fabric operating system service 'binds' and manages the underlying resources at the hosts involved in InfoFabric-based communications. Service morphing, e.g., data filtering and fusing, dynamic filter and handler generation, or service adaptation, is performed on all those components, i.e., in the middleware layer, in the operating system services, in the applications, and in all communication paths between those components.

**Results and status.** The InfoFabric middleware has been used with multiple applications, including scientific applications [31], and elements of the large-scale operational information systems used by companies like Delta Air Lines [6]. To permit precise run-time quality management, a kernel-level implementation of publish/subscribe channels, termed KECho [19], links the operating system kernels of Linux machines that use InfoFabric (or any other) communications. We are able to dynamically deploy handlers 'into' operating system kernels, e.g., to control the monitoring and adaptation actions taken on behalf of applications [12]. KECho has been used for the end-to-end control of interactive video across multiple machines [17], including managing their power consumption [18], and for the online monitoring of distributed resources for scientific applications [12]. Finally, the compiler techniques used when dynamically specializing code, e.g., to match changes in power availability, are currently under development, in part based on our earlier research on runtime code specialization [14] and on code generation of embedded systems [11, 21, 32].

Our research is based on our previous work with distributed, real-time, adaptive, and multimedia systems [24, 27, 30], with sensor-based embedded applications [23], and with high performance middleware [4, 9] and applications [31]. We are also leveraging prior large-scale funding from the National Science Foundation and the Department of Energy that has created the basic publish/subscribe middleware used in this work [4, 33] and is currently creating the new network measurement techniques [10, 22] and middleware mechanisms [7] needed for making middleware 'platform'-aware. Compiler techniques are focused on embedded devices, including static analysis methods like on-chip memory data allocation [28], restructuring for code compaction [11], and efficient data layouts for indirect addressing modes [21]. The InfoFabric project goes beyond such work by using a paradigm of deploying mobile code onto networked devices 'just-in-time' and by proposing notions of dynamic (re)partitioning (using slicing techniques) and its associated dynamic optimizations to generate efficient code for distributed execution platforms.

The following sections describe the components of our approach in more detail.

# 3 InfoFabric

The first component of the service morphing approach described in this paper is the InfoFabric middleware layer. We use the term *InfoFabric*

- to indicate that a distributed source-sink system operates as a whole, where multiple nodes jointly implement complex services provided for multiple end-users, and
- to deal with the high levels of uncertainty on wireless distributed platforms, by casting a 'fabric' of information streams across these systems.

Specifically, the ECho [4] publish/subscribe communication tool targets applications comprised of distributed information sources, transformers, and information sinks. Sources could be web servers, inputs by humans operating electronic devices, or automated sensors that capture information from the physical environment. The information produced by such sources must generally be delivered to multiple sinks, where this information must be transformed, fused, and filtered, so that it arrives in forms useful to end-users. An example is the delivery of client-specific digital data to a large number of remote end-users. These 'services' applied to information flows must be performed within QoS constraints determined not only by data source and data types, but also by current end-user and situational needs. Such constraints must be met continuously, despite changes in the locations and capabilities of the sources, sinks, services, and transports applied to information.

An InfoFabric *service* is comprised of a set of computations applied to information items. These services execute 'underneath' the application-level code that relies on them. Typical services include data filtering, data conversion or encryption, and ancillary computations like those needed for quality of service management. In general, a single service is actually comprised of multiple *code modules* spread across multiple machines, so that both the transport of information and the computations being performed on them are inherently distributed. Since both the information flows in an InfoFabric and the services applied to them are well-specified and known at run-time, the operation of the InfoFabric can be changed dynamically. Such changes are made via run-time adaptations that take advantage of meta-information about typed information flows, information items, services, and code modules in the InfoFabric (i.e., using distributed directory [2] and format [4] servers). Changes involve alterations to services' internal composition, location, and runtime behavior, including dynamically generating, re-generating, and specializing the actual code that implements services. For instance, by employing runtime binary code generation based on precise, compiler-level intermediate specifications of a service's code modules and their internal representations, the service may be quickly (re-)partitioned to match the remaining power budget on an underlying computational node or to match the amounts of information flowing across certain InfoFabric links to the link bandwidths that are currently available [32]. In addition, we can associate *attributes* with event submission and receipt that capture or describe quality of service needs or monitoring data. Such attributes may be used to coordinate adaptation actions across multiple system levels (e.g., middleware and network protocols [7]), or they may be used to enforce desired end-to-end behavior [30].

**Services and meta-information.** Information providers and consumers subscribe to shared logical communication *channels*. Existing Java- or CORBA-based implementations of such publish/subscribe paradigms typically use concentrators to collect and re-send data sent to a channel. In contrast, InfoFabric uses direct source-to-sink links between all providers and consumers of a channel. Furthermore, the information on these channels is represented by *self-describing* information items, where the types of information items being transported are identified to anyone receiving these items and must match the in/out parameter types of the computations performed on these items, thus creating a tight and well-defined linkage of communication with computation. An InfoFabric *service*, then, is defined as a meaningful set of computations applied to information items.

**Example: Operational Information Systems.** Operational Information Systems (OIS) provide continuous support for a company's daily operations on large-scale distributed hardware. An example is the OIS run by Delta Air Lines, which provides the company with up-to-date information about all of its operations, including flights, crews, or passengers. It is driven by the real-time acquisition of data from many disparate, remote sources, such as FAA data feeds and airports. One of the principal tasks is to integrate such data events, by applying to them relevant 'business logic'. The resulting 'business events' have to be made available to a potentially large number of remote subscribers, such as airport flight displays. The model of real-time event subscription, derivation, and publication it thus implements distinguishes an OIS from Data Warehouses and Decision Support Systems: This model is also a basis on which the OIS builds its interactions with other system components, like those participating in E-commerce interactions like passenger reservations. This OIS combines three different sets of functionality:

- *Continuous data capture* – as with high performance applications that capture data from remote instruments like radars or satellites, an OIS must continuously capture all data relevant to an organization's operations.
- *Continuous state updates* – as with digital library servers that must both process newly received data and then distribute it to clients that request it [13], an OIS must both continuously process all operational information that is being captured and then store/distribute the resulting updates of operational state to all interested parties. In the case of Delta Air Lines, this includes low-end devices like airport flight displays, but also large-scale databases in which operational events are recorded for logging purposes. The machine doing this work is termed an Event Derivation Engine (EDE).
- *Short response times* – an OIS must also respond to client requests. In our example, clients request new 'initial states' when airport or gate displays are brought back online after failures, or when agents initialize their displays, and certain clients may generate additional state updates, such as changes in flights, crews, or passengers. To provide timely service to passengers and optimize operational costs, it is important to respond to such requests with predictably bounded delays.

As stated above, the central part of an OIS is its event derivation engine (EDE). For example, Delta uses a cluster of IBM S/390s that run the specialized TPF (Transaction Processing Facility) operating system. Performance requirements of an EDE for operational information systems like Delta's include

- 99.99% average availability;
- performance of up to 5400 transactions per second, with
- 1-3 seconds response time attained 95% of the time;
- at a cost of less than 2 cents per transaction.

Responsiveness requirements include the ability to service requests from clients within bounded time delays not exceeding 1-2 minutes. Sample requests include downloads of new initial states for end users (e.g., airport displays) after failures and updates to gate agents' operating systems.

**InfoFabric results.** One specific technique is *adaptive event mirroring for improved performance and responsiveness*. Events coming into an EDE are mirrored to additional server engines in order to attain low average delays for EDE event derivation and publication, even when the EDE experiences high loads in terms of the number of 'initial state' requests received from recovering clients or from other EDE nodes. Adaptive mirroring means that one dynamically modifies the granularity at which mirroring is performed, thereby also modifying the level of data consistency maintained across different mirror nodes. Such adaptations are made possible by exploiting the streaming nature of the incoming data events and by using certain application-level semantics, such as the knowledge that a position update for a flight will be followed by future position updates, so that it need not be mirrored when loads are high. The idea is to use runtime adaptation to reduce or eliminate the EDE's timing failures [8, 20]. The benefits of adaptive mirroring are explored in more detail in [6], but representative results depicted in Figure 2 trade system consistency vs. QoS by dynamically modifying a mirroring function or its parameters. Useful adaptations strategies include

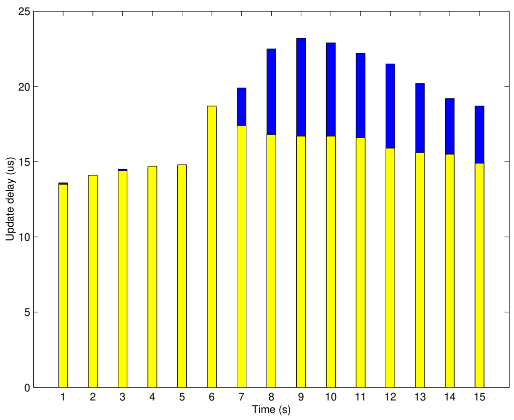


**Figure 2. Adaptive event mirroring.**

- coalescing multiple events vs. mirroring them independently,
- setting the maximum number of events that can be overwritten in a sequence,
- varying checkpointing frequency, and
- installing different mirroring functions.

Two mirroring functions are (a) one that coalesces up to 10 events and then produces one mirror event, thus overwriting up to 10 flight position events, where checkpointing is performed for every 50 events; and (b) one that overwrites up to 20 flight position events and performs checkpointing every 100 events. We compare the EDE execution that adaptively selects which one of these functions to use, depending on the sizes of monitored queues and buffers, with an EDE execution that performs no such runtime adaptation. The performance metric is the processing delay experienced by events from the time they enter the OIS system until the time they are sent to clients by the EDE at the central site. That is, we are evaluating the OIS' contribution to the perturbation experienced by OIS clients receiving state updates. In Figure 2, the darker bars show the increase in average event update delays, which result from an increased rate of incoming client requests. This increase can be avoided by moving request servicing functionality onto mirror nodes. Adaptive mirroring can further modify the mirroring function, reduce its frequency, allow event coalescence, or some filtering based on event type or event content. The lighter bars in the same figure correspond to delays when such adaptation of the mirroring function takes place. Such runtime adaptation has substantial advantages, where total processing latency of the published events is reduced by up to 40%, and the performance levels offered to clients experience much less perturbation than in the non-adaptive case.

## 4 Q-Fabric

Q-Fabric is the kernel-level (currently for the Linux operating system) infrastructure that 'connects' distributed resource monitoring and management agents. These agents communicate and coordinate their actions using the KECho kernel-level event service via *Q-channels* [17]. The architecture of Q-Fabric depicted in Figure 3 shows that applications and kernel-level resource managers share the same Q-channel (i.e., control path), which simplifies the support of the *integrated* management of resources and applications. Note that a Q-channel as depicted in Figure 3 is not a centralized unit, but is distributed among all participating hosts, i.e., all hosts communicate directly with each other (e.g., via socket connections) and channel information (e.g., lists of participating publishers and subscribers) is replicated at each host.

**Q-Fabric results.** A second application domain driving our research is online collaboration, focusing on large-data interactions like remote visualization. Current scientific codes, such as the Terascale Supernova Initiative[1] funded under the DOE SciDAC program, can already generate relevant data at nearly a gigabit per second. Within the next 5 years, these sizes are expected to grow by at least a factor of 10. In order to provide a remote collaborator with an adequate and timely visualization, the data must be carefully staged and transformed to manage bandwidth, latency, and content representation.

[1] http://www.phy.ornl.gov/tsi/

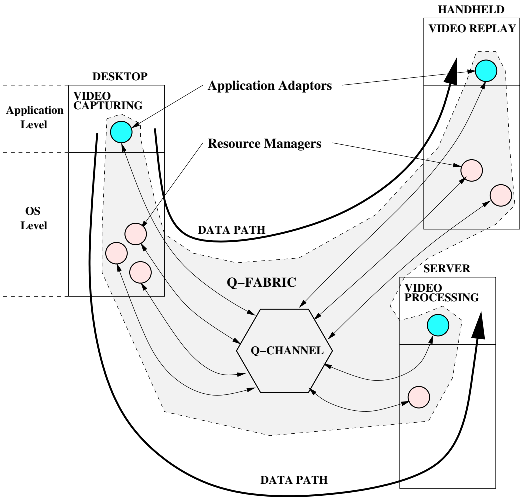


**Figure 3. Q-Fabric architecture.**

In our specific application, a server delivers molecular dynamics data, similar to what is generated by large scientific codes in Physics or Chemistry, to clients that can range from high-end displays like ImmersaDesks to smaller displays like handhelds. The clients can subscribe to any of a number of different derivations of that data, ranging from a straight data feed, to down-sampled data (e.g., removing velocity data), to a stream of images representing the full visualization. Communication is based on InfoFabric's ECho event service; a server establishes an event channel and interested clients can subscribe to this channel in order to receive the data stream. Moreover, clients can customize the data stream by using server-resident *data filters*.

In this scientific visualization example, resource information collected remotely from all clients allows the server to tune the operation of its data filter functions in order to continuously customize the qualities of data streams to the current capabilities of individual clients. The measurements shown in Figure 4 use the monitoring information provided by Q-Fabric to automate the degrees of data filtering based on current resource availability. In other words, the server is made 'aware' of the resources available at different clients via kernel-level runtime monitoring. These results demonstrate two points:

- *system-level support for service morphing*: the importance of information-rich service morphing, where services are changed using the rich and precise resource information available from operating system kernels, and
- *combined compiler- and system-level techniques*: at runtime, data filtering may be tuned not only by setting or re-setting pre-defined parameters, but also by deploying into servers entirely new filters, thus making servers truly 'client-conscious'.

In this experiment, the server sends large events (3MBytes) to clients, and the network link between the client and server is artificially perturbed by running *Iperf*. As network perturbation increases, available bandwidth decreases and latency increases. The capacity of the link is 100Mbps. When there is no perturbation, the server sends data to the client at a rate of about 30Mbps. Hence the plot remains almost horizontal until 70Mbps of perturbation. As perturbation increases beyond 70Mbps, latency drastically increases for the situations where no filtering or static filters – i.e, filters that can not be tuned – are used. However, with dynamic filters, the server dynamically reduces data size in proportion to observed reductions in available network bandwidth, therefore minimizing the effect network perturbation has on the latencies.

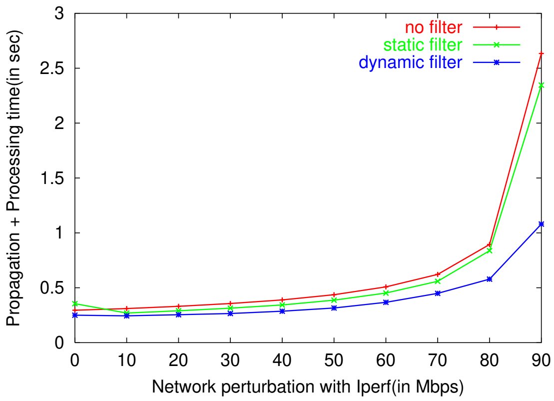


**Figure 4. Client-aware data streaming: change in latency with varying network load.**

Figure 5 shows the change in the event rate at a CPU loaded client. This figure shows that in the dynamic filter case, the client is able to receive and process events at the same rate at which the server sent them. Therefore the inter-event arrival delay remains almost constant. The static filter case cannot adapt itself to the increased load in the system and hence the queuing delays increase and the intervals between event arrivals get larger, although the server is sending these events at a constant rate. The case without filters shows the worst performance.

In the next experiment, Q-Fabric is used to connect the resource managers for a video conferencing application, called *vic*. The resources being managed are:

- **CPU:** The CPU scheduler used in this example is the

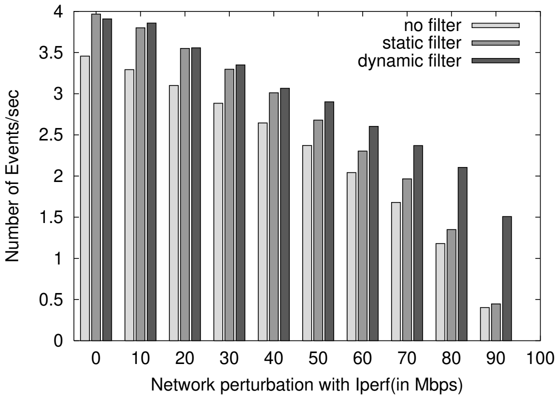


**Figure 5. Client-aware data streaming: change in event rate with varying network load.**

Linux real-time round-robin scheduler. The resource manager adjusts an application's priority class to react to its varying computational needs. Applications are assigned a default priority class (e.g., 50 in the following experiments) and can be modified in the range of 1 and 99.

- **Network:** The Dynamic Window-Constraint Scheduler or DWCS [29] is a real-time scheduler based on three attributes: a period $T$, a window-constraint or loss rate $x/y$, and a run time $c$, where DWCS guarantees an activity $c$ time units of service within a period $T$. However, this guarantee is relaxed by the loss rate, which indicates that $x$ service invocations in $y$ consecutive periods (i.e., $y * T$ time units) can be missed. These attributes translate easily to streaming multimedia applications that require the generation and transmission of data (such as video or audio) with a certain rate. However, such applications can often tolerate infrequent losses or misses of data generation or transmission. If a packet is not scheduled within a period $T$, it is said to have *missed* its deadline. If the number of missed deadlines exceeds $x$ in a window of $y$, the stream is said to have suffered a *violation*. The adjustable parameters of a DWCS stream are the period and the loss-rate. The following experiment uses a default period of $50ms$ (to achieve a frame rate of 20fps) and a loss rate of $x/y = 1/10$.

In addition, the application itself can be adapted by varying the chosen image quality. For the H.261 encoding method, vic supports image qualities in the range of 1 (low quality) to 95 (high quality). The chosen image quality has a significant influence on the processing requirements of the application and the image size transmitted.

This experiment compares the jitter of video replay in vic for three situations:

- without adaptive measures,
- with distributed resource management, and
- with distributed resource management and application adaptation.

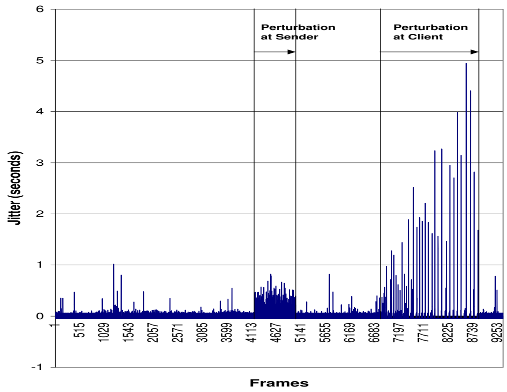


**Figure 6. Jitter in frame replay (client) without adaptation.**

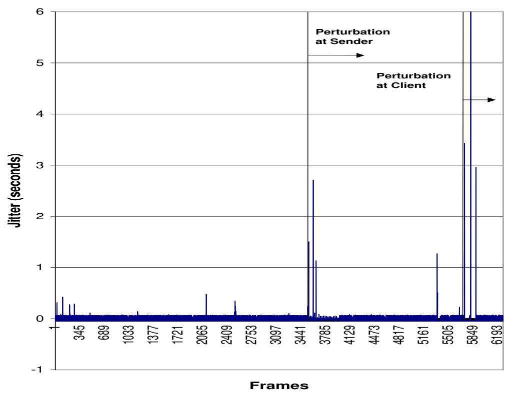


**Figure 7. Jitter in frame replay at client with distributed resource management.**

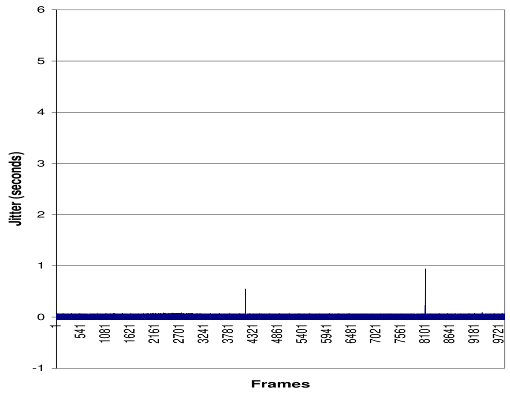


**Figure 8. Jitter (client) with distributed resource management and application adaptation.**

Figure 6 shows the jitter for video replay when no adaptive measures are taken. After approximately 4100 frames, the sender side of the application is perturbed with a CPU-intensive task (endless for-loop), causing a high jitter. After 6800 frames, a perturbation at the client is started, again with a CPU-intensive task, causing a jitter of up to 5 seconds. In Figure 7 we repeat this experiment, however, the Q-Fabric resource management adjusts both network and CPU allocations to the video player at the client. It can be seen that the average jitter is significantly lower during perturbation than in the case without adaptation. However, the jitter still reaches up to 3 seconds for the sender-side perturbation and up to 6 seconds for the client-side perturbation. Finally, in Figure 8, we add application adaptation, i.e., the server-side vic application adjusts the image quality of the H.261 encoded images (between 1 and 30) if the frame rate differs from the desired frame rate. Here, the average jitter is negligible and remains below 1 second in the worst case.

## 5 (Re-)partitioning and (re-)deployment.

The services (i.e., computations) an InfoFabric applies to its information flows are well-defined from the points of view of end users, but their internal composition, location, and runtime behavior are easily and dynamically varied, automatically and often invisibly. Our work goes beyond parameter- or mode-based adaptations of services, by dynamically generating, re-generating, and specializing the actual code that implements services. Specifically, in order to perform fine-grain service re-location and re-partitioning, we maintain meta-information about each service's code modules, including their locations, their relationships to each other, and their internal structures, using distributed directory [2] and format [4] servers. In addition, with InfoFabric event submission or receipt, we can associate *attributes* that capture or describe quality of service needs and/or monitoring data. Such attributes may be used to coordinate adaptation actions across multiple system levels (e.g., middleware and network protocols [7]), or they may be used to enforce desired end-to-end behavior [30]. In general, such meta-information can be sufficiently detailed to not only enable the run-time re-deployment of a service's statically defined code modules, but also to 'take apart' and 're-assemble' code modules to better match an information flow's transport and operation to the current capabilities of the underlying embedded system (see [32] for a detailed description of results attained with a Java implementation of ECho, termed JECho). The resulting 'lightweight' dynamic service re-partitioning methods will enable InfoFabric applications to operate with degrees of flexibility akin to those of Java-based systems. For instance, by employing runtime binary code generation based on precise, compiler-level intermediate specifications of a service's code modules and their internal representations, the service may be quickly re-partitioned to match the remaining power budget on an underlying computational node or to match the amounts of information flowing across certain InfoFabric links to the link bandwidths that are currently available.

## 6 Conclusions and Future Work

The computing infrastructures being developed for distributed autonomic systems will have to have several of the capabilities offered by our approach. Agility in terms of rapid code deployment and re-deployment is required to match program code to platforms. Adaptivity is required to match how services are run and applied to data to create time-critical information for end-users with the quality they need. Methods for dealing with rapid infrastructure changes, especially in critical systems, not only have to recover from such changes but should also adjust middleware-level actions and behavior to such changes. Some of those methods require kernel-level support to be able to operate at all or at the high levels of granularity required by applications.

Our future work will continue to use multiple applications, including online collaborations involving large data volumes, the rapid access to remote live (i.e., sensor) data, and commercial applications like operational information systems. We will improve application performance by developing distributed algorithms that change the ways in which information is routed and processed across a distributed embedded platform, resulting in *overlay networks* [1, 3]. Novel 'in place' and 'remote' compilation techniques will implement dynamic service morphing, by generating new code to be deployed at certain overlay network nodes. Middleware and code generation will not only enable a component of an InfoFabric service to be dynamically re-deployed, but also to be 'taken apart', 'reassembled', and specialized to better match an information flow's transport and operation to the current capabilities of the underlying system. The lightweight dynamic service (re-)partitioning methods resulting from this research will enable embedded and distributed systems software to operate with degrees of flexibility akin to those of Java-based systems.

## References


[1] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[2] F. Bustamante, P. Widener, and K. Schwan. The case for proactive directory services. In *Proc. of Supercomputing 2001 (Poster), Denver, CO*, November 2001.

[3] Y. Chen, K. Schwan, and D. Zhou. Opportunistic channels: Mobility-aware event delivery. In *Proc. of the ACM/IFIP/USENIX Middleware 2003 Conference*, 2003.

[4] G. Eisenhauer, F. Bustamente, and K. Schwan. Event services for high performance computing. In *Proceedings of High Performance Distributed Computing (HPDC-2000)*, 2000.

[5] A. Gavrilovska, K. Mackenzie, K. Schwan, and A. McDonald. Stream handlers: Application-specific message services on attached network processors. In *Proceedings of the Tenth Symposium on High Performance Interconnects (HOT-I 2002), Palo Alto, CA*, August 2002.

[6] A. Gavrilovska, K. Schwan, and V. Oleson. Adaptable mirroring in cluster servers. In *Proc. of High Performance Distributed Computing (HPDC) Conference, San Francisco, CA*, August 2001.

[7] Q. He and K. Schwan. Iq-rudp: Coordinating application adaptation with network transport. In *Proc. of High Performance Distributed Computing*, 2002.

[8] M. A. Hiltunen, V. Immanuel, and R. D. Schlichting. Supporting customized failure models for distributed services. *Distributed System Engineering*, Dec. 1999.

[9] C. Isert and K. Schwan. Acds: Adapting computational data streams for high performance. In *Proceedings of the 2000 International Parallel and Distributed Processing Symposium (IPDPS)*, May 2000.

[10] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *Proc. of the ACM SIGCOMM 2002 Conference, Pittsburgh, PA*, August 2002.

[11] V. Jain, S. Rele, S. Pande, and J. Ramanujam. Code restructuring for improving real time response through code speed, size trade-offs on limited memory embedded dsps. *IEEE Transactions on CAD*, 20(4):477–494, April 2001.

[12] J. Jancic, C. Poellabauer, K. Schwan, M. Wolf, and N. Bright. dproc - extensible run-time resource monitoring for cluster applications. In *Proc. of the International Conference on Computational Science*, 2002.

[13] R. E. McGrath, J. Futrelle, R. Plante, and D. Guillaume. Digital library technology for locating and accessing scientific data. In *ACM Digital Libraries '99*, August 1999.

[14] D. McNamee, J. Walpole, C. Pu, C. Cowan, C. Krasic, A. Goel, P. Wagle, C. Consel, G. Muller, and R. Marlet. Specialization tools and techniques for systematic optimization of system software. *ACM Transactions on Computer Systems*, 19(2):217–251, May 2001.

[15] V. Oleson, K. Schwan, G. Eisenhauer, B. Plale, C. Pu, and D. Amin. Operational information systems - an example from the airline industry. In *Proceedings of the First Workshop on Industrial Experiences with Systems Software (WIESS)*, 2000.

[16] A. Orso, A. Rao, and M. Harrold. A technique for dynamic updating of java software. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2002)*, October 2002.

[17] C. Poellabauer and K. Schwan. Kernel support for the event-based cooperation of distributed resource managers. In *Proc. of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2002), San Jose, CA*, September 2002.

[18] C. Poellabauer and K. Schwan. Power-aware video decoding using real-time event handlers. In *Proceedings of the 5th International Workshop on Wireless Mobile Multimedia (WoWMoM)*, September 2002.

[19] C. Poellabauer, K. Schwan, G. Eisenhauer, and J. Kong. Kecho - event communication for distributed kernel services. In *Proc. of the International Conference on Architecture of Computing Systems (ARCS'02), Karlsruhe, Germany*, April 2002.

[20] D. Powell. Failure mode assumptions and assumption coverage. In *Proc. of the 22nd Symposium of Fault-Tolerant Computing*, 1992.

[21] A. Rao and S. Pande. Storage assignment optimizations to generate compact and efficient code on embedded dsps. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '99)*, pages 128–138, June 1999.

[22] N. S. V. Rao, S. Radhakrishnan, and B. Y. Cheol. Netlets: Measurement-based routing for end-to-end performance over the internet. In *Proc. of the Intl. Conference on Networking*, 2001.

[23] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. On adaptive resource allocation for complex real-time applications. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS)*, December 1997.

[24] D. I. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. On adaptive resource allocation for complex real-time applications. In *18th IEEE Real-Time Systems Symposium, San Francisco, CA*. IEEE, Dec. 1997.

[25] L. Sha, X. Liu, and T. Abdelzaher. Queuing model based network server performance control. In *Proc. of the Real-Time Systems Symposium*, Dec. 2002.

[26] 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.

[27] 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.

[28] W. Tembe, L. Wang, and S. Pande. Data I/O minimization for loops on limited on-chip memory embedded processors. to appear in IEEE Transactions on Computers.

[29] 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.

[30] R. West and K. Schwan. Interactors: Capturing QoS and Resource Requirements Between Multiple Cooperating Objects. In *Fourth IEEE Real-Time Technology and Applications Symposium, Work-In-Progress*. IEEE, 1998.

[31] M. Wolf, Z. Cai, W. Huang, and K. Schwan. Smartpointers: Personalized scientific data portals in your hand. In *Proc. of SuperComputing 2002*, Nov 2002.

[32] D. Zhou, S. Pande, and K. Schwan. Method partitioning - runtime customization of pervasive programs without design-time application knowledge. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, May 2003.

[33] 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.

[34] 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. `http://www.dist-systems.bbn.com/tech/QuO`.