Adaptable Mirroring in Cluster Servers

Ada Gavrilovska, Karsten Schwan, Van Oleson

GIT-CC-01-01

College of Computing
Georgia Institute of Technology
Atlanta, GA, 30329
{ada, schwan, van}@cc.gatech.edu

Abstract

This paper presents a software architecture for continuously mirroring streaming data received by one node of a cluster-based server system to other cluster nodes. The intent is to distribute the server loads implied by the data's processing and distribution to large numbers of clients. This is particularly important when a server performs multiple processing tasks, some of which may heavily depend on current system status. One specific example is the preparation of suitable initialization state for thin clients, so that such clients can understand future data events being streamed to them. In particular, when large numbers of thin clients must be initialized at the same time, such initialization must be performed without jeopardizing the quality of service offered to regular clients continuing to receive data streams.

The mirroring framework presented and evaluated here has several novel aspects. First, by performing mirroring at the middleware level, application semantics may be used to reduce mirroring traffic: by event filtering based on data type or even data content, by coalescing certain events, or by simply varying mirroring rates according to current application needs concerning the consistencies of mirrored vs. original data. The intent of such dynamically varied mirroring is to improve server scalability, both with respect to its ability to stream data events to a large number of clients and to deal with large and highly variable request volumes from clients that require other services, such as new initial states computed from incoming data events. Second, this paper presents an adaptive algorithm that varies mirror consistency and thereby, mirroring overheads in response to changes in clients' request behavior. The third, novel aspect of this work is the framework's ability to not only mirror events, but to also mirror the new states computed from incoming events, thus enabling dynamic tradeoffs in communication vs. computation loads imposed on the server node receiving events and on its mirror nodes. This framework capability is used for adaptive event coalescing in response to increases or decreases in client request loads. Fourth, mirroring functionality is structured so that it is easily 'split' across nodes' main CPUs and the CoProcessors resident on their programmable network interfaces (should such interfaces exist). A case study of a mirroring design for an Intel I2O-based interface board using an i960 RD CoProcessor is also presented in this paper.

1 Operational Data Services and Servers

Server systems structured as cluster machines have become increasingly common, to drive search engines [1], operate mail servers [2], or provide scientific data or computational services [3, 4]. Our research concerns one important property of such cluster servers, which is their ability to continue to provide high levels of service even under increasing loads or when clients' request behaviors vary dynamically.

Operational Information Systems. In contrast to the interactive high performance applications considered in our other research [5, 6], this paper addresses an emerging application domain for high performance computing and for the cluster servers on which these applications run, which is that of operational information systems [7] (OIS). An OIS is a large-scale, distributed system that provides continuous support for a company's or organization's daily operations. One example of such a system we have been studing is the OIS run by Delta Air Lines, which provides the company with uptodate information about all of their flight operations, including crews, passenger, and baggage. Delta's 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, or from physical instruments like the particle accelerators described in [8], an OIS must continuously capture all data relevant to an organization's operations. For airline operations, such information includes crew dispositions, passengers, airplanes and their current locations determined from radar data available from the FAA, etc.
- *Continuous state updates* – as with digital library servers that must both process newly received data (e.g., NASA's satellite data) and then distribute it to clients that request it [9], an OIS must both continuously process all operational information that is being captured (i.e., by applying relevant 'business logic' to it) 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, the PCs used by gate agents, and even large-scale databases in which operational state changes are recorded for logging purposes.
- *Response to client requests* – an OIS not only captures data and updates/distributes operational state, but it must also respond to explicit client requests. In the case of Delta's OIS, clients request new initial states when airport or gate displays are brought back online after failures, or when agents initialize their displays (e.g., agent login), and certain clients may generate additional state updates, such as changes in flights, crews, or passengers.

Problem: predictable performance with bursty requests. The problem addressed by this paper is how to offer predictable performance in an operational system, in the presence of bursty client requests. Stated more explicitly, for an OIS this means that (1) the central server must continue to process captured data and update operational state, (2) the server must continue to distribute state updates to clients, and (3) it must respond to explicit client requests in a timely fashion. In this example, predictability can be quantified precisely: Delta Air Lines currently requires that no essential operational state is lost, that the distribution of state updates be performed within less than 30 seconds after they have been received, and that client requests like those for state initialization must be satisfied within less than a minute. In general, the timeliness requirements concerning responses to client requests and concerning ongoing operational state capture and update vary across applications. For multimedia applications, for instance, clients may be sent lossy data. For applications like Delta's OIS, data cannot be lost, and acceptable delays are determined by the business processes being applied to operational data and by the need to offer what appears to be 'zero' downtime and consistent flight information to gate agents' and passengers' views of operational data. For applications like IBM's information services for the Atlanta Olympic Games, even small delays were devastating: both television viewers and journalists were disappointed when IBM's servers could not keep up with the bursty volume of requests for updates while also collecting and collating the results of recent sports events in a timely fashion.

Solution approach: adaptable mirroring in cluster servers. This paper explores the following approach to achieving predictable performance in large-scale data servers like Delta's OIS: (1) we use cluster machines as data servers, which internally use high bandwidth, low latency network connections and can be connected to the remainder of the OIS via multiple network links, and (2) we devise mechanisms and policies to dynamically 'spread' the operational state being computed across the cluster machine, as well as distribute client requests across the cluster. The intent is to continuously provide timely service for both operational data processing and client requests. The technical approach implemented and evaluated in this paper is that of *data mirroring*, where captured data structured as application-level *update events* is mirrored to all other cluster machines interested in such events. For such update events, all mirror machines execute the same business logic, thus resulting in the OIS's operational state being naturally replicated across all cluster machines participating in event mirroring. The result of such mirroring is that clients' requests for OIS state may be satisfied not just by one, but by any one of the mirror machines. The resulting parallelization of request processing for clients coupled with simple load balancing strategies enables us to offer timely services to clients even when request loads become high.

Mirroring of update events and the resulting load balancing of client requests are make feasible in part by the architecture of the cluster computer used for implementing OIS. This is because intra-cluster communication bandwidth and latency are far superior to those experienced by data providers and by clients. Furthermore, we share the assumption with other cluster-based server systems [2] that OIS software is structured such that all cluster machines can perform all OIS tasks. This assumption is reasonable since it is not the complexity of OIS 'business logic' that causes performance concerns but instead, performance issues arise due the high rates of incoming data and the large number of clients for operational data (numbering in the 10's of thousands for Delta's OIS, for example). Finally, we do not assume that all cluster machines act as mirrors, and that all OIS state is replicated. Instead, since OIS state can grow to many gigabytes, in general, update events must be mirrored both to sites that replicate local state and to sites that need such events for functionally different tasks. This also implies that in general, client requests must be distributed both according to the functional distribution of OIS software on the cluster and the mirroring of state that is being performed. This paper ignores the functional distribution of OIS state and instead, focuses on event mirroring, its performance implications, and its utility.

Advantages of mirroring. We seek two advantages from data mirroring: (1) by distributing client requests across mirrors, request loads, especially under high load conditions, may be distributed across multiple cluster machines, thus responding to client requests in a timely fashion under variable load conditions, and (2) by reducing the perturbation imposed on the OIS from bursty client requests, the predictability with which the OIS server can continuously capture, process, and deliver operational information is improved. Results presented in Section 3 demonstrate, for instance, that a cluster machine can mirror update events to a moderate number of cluster machines without imposing undue loads on the cluster interconnect, and that the perturbation caused by client requests can be controlled in this fashion. In addition, prior work has established that simple request load balancing algorithms offer good performance on cluster server machines [1, 10]. Another, well-known reason for data mirroring is the increased reliability gained from the availability of critical data on multiple cluster nodes [11], a topic not explored in detail herein.

Performance issues and solutions. The basic technical problem we address for cluster servers used in operational information systems is their scalability in face of dynamic variations in client request patterns and/or of the captured information being processed. The scalability metric we use attempts to capture predictability of the services implemented by the server: how does a server react to additional loads it experiences, with respect to deviations in the levels of service offered to its 'regular' clients? With this metric, high server scalability implies small deviations for large additional loads. The intent of this metric is simple. It measures how an operational information system reacts to unusual or irregular operating conditions, with respect to its continued ability to provide information to its clients at the rates and with the latencies dictated by ongoing business processes. In the case of an airline, unusual operating conditions include (1) 'bringing up' an airport terminal after a power failure, and (2) dealing with inclement weather conditions. Case (1) requires the airport terminal's many 'thin clients' (e.g., airport displays) to be re-supplied quickly with suitable initial states, thereby once again enabling them to interpret the regular flow of data events issued by the server. Case (2) results in changes in the distributions of captured events and therefore, changes in the loads imposed on certain service tasks. For instance, in inclement weather conditions, it would be appropriate to track planes at increased levels of precision, thus resulting in increased loads on servers caused by the additional tracking processing and in increased communication loads due to the distribution of tracking data.

Both Cases (1) and (2) have similar effects on the data server: when faced with these additional loads, it cannot continue to issue output events at the rates required by its clients. Experiments shown in Section 3 demonstrate this fact. However, these experiments also demonstrate problems that arise from our approach of event mirroring. These problems are that the overheads of event mirroring itself can be significant, as well, resulting in a 30% slowdown on our cluster machine when there are 4 mirror machines, for instance. The remainder of this paper, then, investigates technical solutions to address these problems and thereby establish the practicality of event mirroring:

- *Mirroring overheads* – the intent of mirroring is to improve server scalability, in part by offloading tasks from certain server nodes. Yet, at the same time, mirroring overheads can slow down server nodes and thereby reduce scalability. The performance effects of mirroring experienced on a cluster machine are explored in detail in Section 3.1.
- *Application-specific mirroring* – mirroring differs from message broadcast or multicast in that it is performed at the application level. This enables us to substantially reduce mirroring traffic compared to implementations described previously [11], by filtering events based on their data types [12] and/or their data contents [13], by coalescing certain events particularly when the effects of a later event overwrites those of previous events, or by simply varying mirroring rates according to current application needs concerning the consistency of mirrored vs. original data. In Section 3.2, we experimentally evaluate the performance implications of some of these choices.

- *Adaptive mirroring* – overheads in mirroring are due to the rates at which data events are mirrored and the degrees of data consistency across multiple cluster nodes enforced by the mirroring mechanism. Thus, by reducing consistency or rates when servers experience increased request loads due to unusual operating conditions, server scalability can be improved substantially. We also present a dynamic algorithm for mirroring adaptation and evaluate its impacts on server scalability in Section 3.3.

Remainder of paper. In the remainder, of this paper, we first explain in more detail the OIS application that drives our research. Next, we briefly outline a software architecture for event mirroring, present some implementation detail, and then describe experimental results attained on cluster machines at Georgia Tech.

2 The OIS Application and Event Mirroring

2.1 The OIS Application

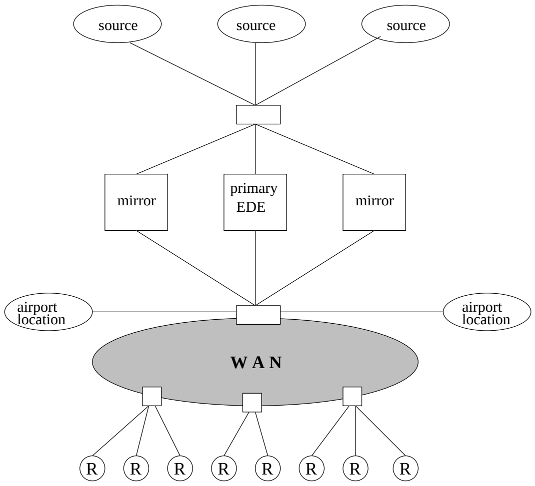

Figure 1: Delta application

Figure 1 shows how operational information comprised of update events describing airplanes' locations and crew availability, for instance, is captured by a wide area collection infrastructure. This results in a continuous stream of update events received by the OIS server. The data events being captured from, processed, and distributed to large numbers of company sites concern the current state of company assets (e.g., airplanes or crews) and that of their customers' needs (e.g., passenger or luggage volumes and destinations). The ability to track and manipulate such information in a timely fashion is critical to the company's operations and profits. In the case of Delta Air Lines, events are captured at system entry points like gate readers or radar data collected by the FAA, distributed via company-internal or leased network links, and processed by servers located at the company's processing center in Atlanta. The outputs generated by the center's server are used by a myriad of clients, ranging from simple airport flight displays to complex web-based reservation systems. A representative processing action performed by the OIS server is one that determines from multiple events received from gate readers that all passengers of a flight have boarded.

The code executed by the OIS server is what we term its Event Derivation Engine (EDE). EDE code (1) performs transactional and analytical processing of newly arrived data event, according to a set of business rules, and (2) it services incoming clients' requests. More specifically, it provides clients with initial views of the states of operational data on demand. Once they receive this initial views, clients maintain their own local views of the system's state, which they continuously update based on events received from the OIS server.

2.2 Event Mirroring – Software Architecture and Implementation

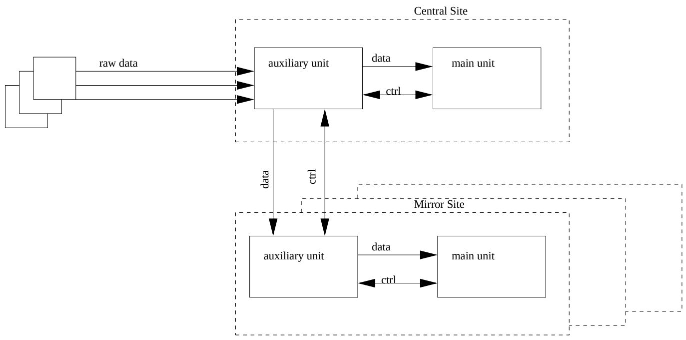

Figure 2: Architecture of the mirroring framework

Basic architecture. The prototype architecture of our mirroring framework is depicted in Figure 2. Events may be provided by any number of data sources, and they are processed by a 'central' site and its set of mirror machines. In our current implementation, a single central site located on the cluster machine acts as the primary mirror, with other sites acting as secondary mirrors. In an actual, deployed operational system, it should be possible to separate data mirroring from processing functionality, thereby permitting us to use application-specific extensions [14, 15] of operating system kernels of communication co-processors (i.e., network interface boards) to reduce mirroring overheads. The resulting software architecture used for data mirroring, then, separates the application-specific code (i.e., business logic) executed by the Event Derivation Engine from the code that implements mirroring (labelled as 'main' and 'auxiliary' units in the figure). Events produced by each mirror site are typically generated by business logic, in response to receiving multiple incoming update events. All mirrors produce the same output events, and produce identical modifications to their locally maintained application states.

The main functionality of the central site is distribution of the continuous state updates, while the mirror sites are mainly involved in providing responses to explicit clients' requests. Three separate tasks execute within the auxiliary unit of the central site: (1) a receiving task, (2) a sending task, and (3) a control task. Their execution is synchronized via accesses to common data queues, which include the *ready* and the *backup* data queue, a *status table* containing relevant status information for application-level processes (e.g., flight status), and a set of paramenters determined by the application and embodying the *semantic rules* that are to be followed by the mirroring process. A sample semantic rule is one that tells the mirroring process to eliminate certain incoming events when mirroring them (examples appear below). The receiving task retrieves events from the incoming data streams, performs the timestamping and event conversion when necessary, and places the resulting events into the ready queue. Events are removed from the ready queue, sent onto all outgoing channels, and temporarily stored in the backup queue by the sending task. The control task runs various control routines, including (1) checkpointing, upon which all successfully checkpointed events are removed from the backup queue, and (2) the exchange of values of monitored variables and new settings for the mirroring parameters during the adaptation process, as explained further in Section 3. Additional control tasks under development include recovery in the presence of illformed communications or failures in EDE processing.

Implementation. We use the ECho event communication infrastructure [6] to efficiently move data across machines. Thus, communication is carried out via multiple logical event channels, both between mirrors and among sources, mirrors, and clients. Two separate event channels exist between the auxiliary and the main unit on each site, as well as between the two auxiliary units: a 'data' channel carries application-specific information, captured in data events, and a bi-directional 'control' channel carries control events, necessary for guaranteeing consistent state across mirrors. Control channels are also used to make changes to the ways in which mirroring is performed.

Two types of event streams exist in our application: one that carries FAA flight position information, and another that carries Delta's internal flight information, such as current flight status (landed, taxiing), passenger and baggage information, etc. For both channels, events themselves are uniquely timestamped when they enter the 'central' mirror site. If an incoming event to be mirrored already has the appropriate format, then no mirror-related processing is necessary in the 'auxiliary unit'. That may not be the case for the 'application-specific' and 'adaptive mirroring' demonstrated in Section 3, where incoming data is first extracted from the event stream, then filtered, and then converted into the appropriate outgoing event format.

3 Experimental Results

Experiments with data mirroring first diagnose the resulting overheads encountered during OIS execution. Adaptive mirroring is introduced to reduce these overheads. Toward this end, we first evaluate the benefits of using certain application-specific information to reduce mirroring traffic and frequency, including the effects of this approach on sustainable request loads and on the update delays experienced by the application. Finally, we analyze the performance implications of dynamic modifications to the mirroring function on overall application performance, in terms of the predictability of the service levels offered to the clients.

Experiments are performed with a mirrored server running on up to eight nodes of a cluster of 300MHz Pentium III dual-processor servers running Solaris 5.5.1. The 'flight positions' data stream used in these experiments originates from a demo replay of original FAA streams, and it contains 100,000 flight position entries for 50 different flights. The evaluation metric is the total execution time of the simulation. To simulate client requests that add load to the server's sites, we use httperf version 0.8, a standard tool for generating HTTP traffic and measuring server performance [16]. Httperf clients run on 550MHz Pentium III Xeon nodes connected to the server cluster via 100Mbps ethernet.

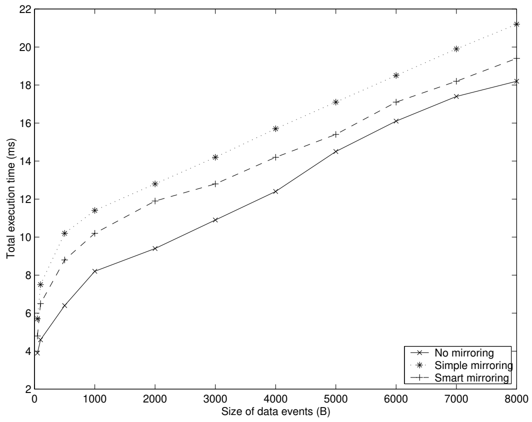

Figure 3: Overhead of mirroring to a single site with 'simple' and 'smart' mirroring

3.1 Mirroring Overheads

A set of microbenchmarks measures mirroring overheads as functions of the data size and the number of mirror sites. No additional load is assumed at any of these sites as a result of incoming client requests. The predictability of the server's execution is observed throughout the microbenchmarks' duration. The experiment terminates after the entire sequence of events presented to the mirroring system has been processed.

Results presented in Fig. 3 indicate that an approximately 15-20% increase in execution time is experienced when all incoming events are mirrored to a single mirror site, the overhead being greater for larger event sizes. This increase is due to event resubmission, thread scheduling, queue management and execution of the control mechanism. Overheads can be reduced by replacing the simple mirroring function with one that mirrors events selectively. In the case of FAA data, when using a selective mirror function that mirrors only the most recent event in a sequence of up to 20 overwriting events, overhead is reduced significantly, with reductions again becoming more pronounced for larger event sizes (dashed line in Fig. 3).

The next set of measurements diagnose the effects of increasing the number of mirrors, while maintaining constant data sizes for the events being mirrored. Our results indicate that on the average, there is a less than 10% increase in the execution time of the application when a new mirror site is added (see Fig. 4).

3.2 Application-Specific Mirroring

Next, we investigate our infrastructure's impact on server performance and scalability, by analyzing the performance of a mirrored server under varying client requests loads, using different mirroring functions. Each experiment compares the total processing time for the same event sequence, for mirrored servers with one, two and four mirror sites, respectively. In all cases, we operate under a constant request load evenly distributed across mirror sites. Our results presented in Fig. 5 consider the total time taken for both processing the entire event sequence and also servicing all client requests. The results show that for data sizes larger then some cross-over size (where experimental lines intersect) the mirroring

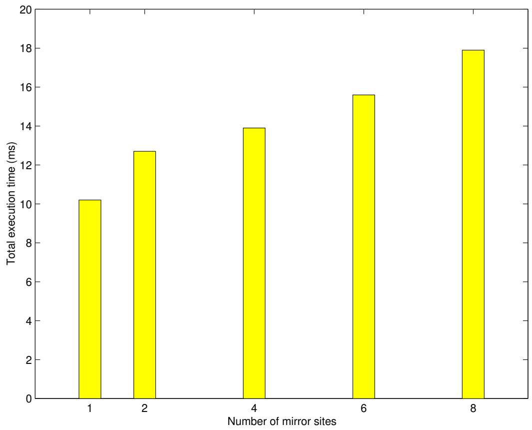

Figure 4: Overhead of adding additional mirrors

overheads can be outweighted by the performance improvements attained from mirroring. Smaller total execution times also indicate that the 'regular' state updates issued by the server are executed more promptly. This indicates the improved server predictability we are seeking to attain by data mirroring.

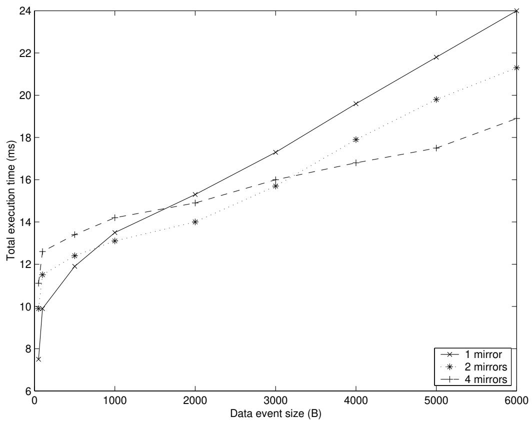

Figure 5: Mirroring to multiple mirror sites, under constant request load of 100 req/sec ballanced across the mirrors.

Server scalability may be improved by using application-specific mirroring function that performs small amounts of additional event processing in order to filter out some of the events, thereby reducing total mirroring traffic. For varying request loads and for one mirror, we compare the use of a simple vs. a 'smart' mirroring function. Results indicate that smart mirroring can improve performance by more than 30% under high request loads (dotted line in Fig. 6). Further adjustments of other mirroring parameters can result in additional improvements. For instance, by decreasing the checkpointing freqency by 50%, total execution time is reduced by another 10%, resulting in a total reduction of more than 40% in the total time required to both process our event sequence and service some fixed, total number of client requests. This is shown with the dashed line in Fig. 6.

Smart mirroring also affects the update delays experienced by operational data clients. Experimental results suggest that the experienced 40% reduction in the total execution time corresponds to a decrease in the average update delay experienced by clients of more than 50%, as presented in Fig. 7.

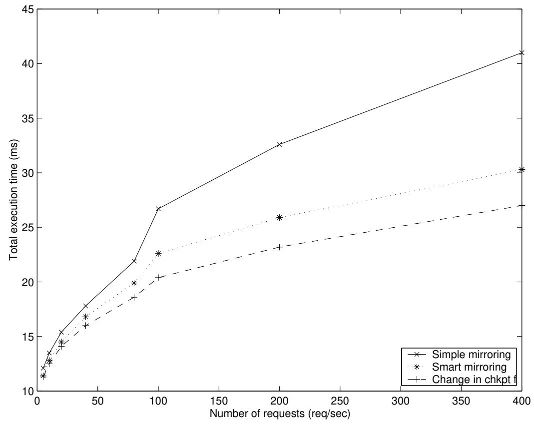

Figure 6: Comparison of three mirroring functions: 'simple', 'smart', and 'smart' with decreased checkpointing frequency

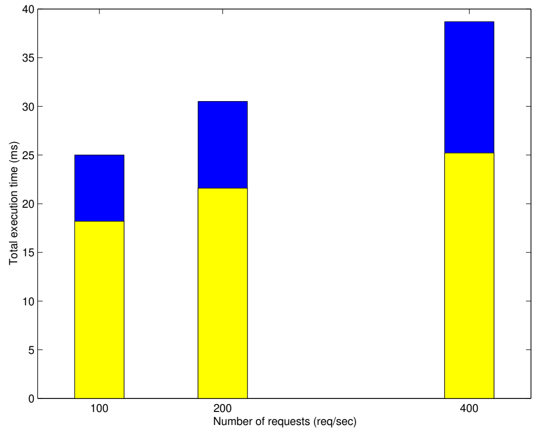

Figure 7: Update delays with 'smart' vs. 'simple' mirroring

3.3 Adaptive Mirroring

The experiments described in Section 3.2 suggest that there is no single 'best' strategy for mirroring events under certain client loads. Instead, such mirroring should be adjusted dynamically, as client request loads increase or decrease, thus continuously adjusting the mirroring system to provide suitable OIS services while also responding to clients in a timely fashion.

We implement adaptive mirroring by continually monitoring the lengths of the ready and backup queues, as well as the sizes of the application-level buffers of pending clients requests. Based on these monitored values and usinga set of predetermined threshold values, we alternately use two mirroring functions. The first one collates up to 10 events and then produces one mirror event, thus overwriting up to 10 flight position events. Checkpointing is performed for every 50 events. The second function overwrites up to 20 flight position events and performs checkpointing every 100 events. We then compare the OIS execution that adaptively selects which one of these functions to use, depending on the sizes of monitored queues and buffers, with an OIS execution that performs no such runtime adaptation. Comparisons are performed with bursty clients requests pattern. 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. In other words, we are evaluating the OIS' contribution to the perturbation experienced by OIS clients receiving state updates.

The results in Fig. 8 show the benefits of even the simple method of runtime adaptation described above. Specifically, total processing latency of the published events is reduced by 40%, and the performance levels offered to clients experience much less perturbation than in the non-adaptive case.

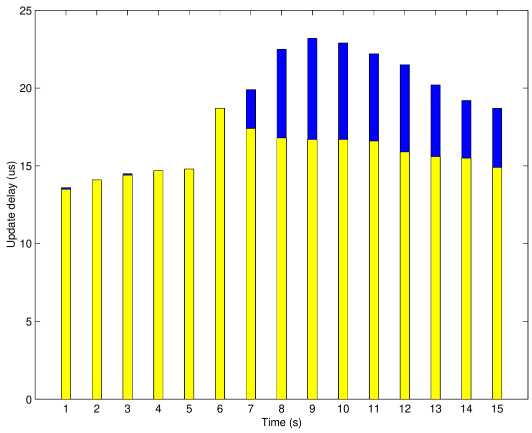

Figure 8: Performance implications of dynamic adaptation of the mirroring function based on the current operating conditions

4 Conclusion and Future Work

This paper is concerned with Operational Information Systems. The scalable construction of such systems on cluster machines is facilitated by what we term *data mirroring*. Data mirroring differs from multicast or broadcast functionality offered at the network level by its use of application-level code to decide what data to mirror and how such mirroring should be performed. The resulting application-level approach to data mirroring is shown to have two benefits on cluster machines that run OIS applications: (1) the continuous processing of incoming data events and subsequent generation of outgoing events is more easily guaranteed when unusual or bursty client requests are directed to mirror sites, rather than to the cluster node(s) that perform OIS processing, and (2) mirroring can help deal with the bursty nature of such requests, by effectively parallelizing client request processing.

Experimental results show that while event mirroring results in additional overheads imposed on OIS processing, such overheads can be reduced by use of application-specific rules that determine what is being mirrored and how frequently mirroring is performed. By dynamically adapting mirroring while monitoring client request patterns, OIS execution can be made more predictable, and clients experience more timely responses.

References

[1] Armando Fox, Steven Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier. Cluster-based Scalable

Network Services. In *Sixteenth ACM Symposium on Operating System Principles*, Oct.1997.

[2] Saito, Bershad, Levy. Manageability, Availability, and Performance in Porcupine: A Highly Scalable Cluster-based Mail Service. In *17th ACM Symposium on Operating System Principles*, OS Review, Volume 33, Number 5, Dec. 1999.

[3] T. Sterling, D. Becker, D. Savarese, et al. BEOWULF: A Parallel Workstation for Scientific Computation. In *Proceedings of the 1995 International Conference on Parallel Processing (ICPP)*, August 1995, Vol. 1, pp. 11-14.

[4] Michael Beynon, Tahsin Kurc, Alan Sussman, Joel Saltz. Optimizing Retrieval and Processing of Multi-dimensional Scientific Datasets. In *Proceedings of the Third Merged IPPS/SPDP Symposiums*, Cancun, Mexico, May 2000.

[5] Beth Plale, Volker Elling, Greg Eisenhauer, Karsten Schwan, Davis King, and Vernard Martin. Realizing Distributed Computational Laboratories. *The International Journal of Parallel and Distributed Systems and Networks*, Vol 2, Num 3, 1999.

[6] Greg Eisenhauer, Fabian Bustamente, and Karsten Schwan. Event Services for High Performance Computing. In *Proc. of Ninth High Performance Distributed Computing (HPDC-9)*, August 2000.

[7] V.Oleson, K.Schwan, G.Eisenhauer, B.Plale, C.Pu, D.Amin. Operational Information Systems - An Example from the Airline Industry. *First Workshop on Industrial Experiences with Systems Software WIESS 2000*, October 2000

[8] J. Becla and A. Hanushevsky. Creating Large Scale Database Servers. In *Proc. of Ninth Symposium on High Performance Distributed Computing (HPDC-9)*, Pittsburgh, Aug. 2000.

[9] Robert E. McGrath, Joe Futrelle, Ray Plante, Damien Guillaume. Digital Library Technology for Locating and Accessing Scientific Data. *ACM Digital Libraries '99*, August, 1999, pp. 188-194.

[10] H. Zhu, T. Yang, Q. Zheng, D. Watson, O.H. Ibarra, and T. Smith. Adaptive Load Sharing for Clustered Digital Library Servers. In *Proc. of Seventh Symposium on High Performance Distributed Computing (HPDC-7)*, Chicago, Aug. 1998.

[11] Kenneth Birman, Andre' Schiper, and Pat Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems*, Aug. 1991, 9, 3.

[12] Dong Zhou and Karsten Schwan. Adaptation and Specialization in Mobile Agent Environments. In *Proc of 5th Usenix Conference on Object-Oriented Technologies and Systems*, San Diego, May 1999.

[13] Carsten Isert and Karsten Schwan. ACDS: Adapting Computational Data Streams for High Performance. In *Proc. of International Parallel and Distributed Processing Symposium (IPDPS)*, May 2000.

[14] Marcel Catalin Rosu, Karsten Schwan, and Richard Fujimoto. Supporting Parallel Applications on Clusters of Workstations: The Intelligent Network Interface Approach. In *Proc of Sixth Symposium on High Performance Distributed Computing (HPDC-6)*, Portland, Aug. 1997.

[15] Christian Poellabauer, Karsten Schwan, Richard West, Ivan Ganev, Neil Bright and Gregory Losik. Flexible User/Kernel Communication for Real-Time Applications in ELinux. In *Proc. of the Workshop on Real Time Operating Systems and Applications and Second Real Time Linux Workshop*, 2000 (in conjunction with RTSS 2000).

[16] David Mosberger and Tai Jin. httperf—A Tool for Measuring Web Server Performance. *Workshop on Internet Server Performance'98*, Madison, June 1998.