Real-Time Cooperative Editing on the Internet

Yun Yang
Swinburne University of Technology
Chengzheng Sun
Griffith University
Yanchun Zhang
University of Southern Queensland
Xiaohua Jia
City University of Hong Kong

New research and a prototype implementation solve some fundamental problems in distributed, real-time, cooperative editing in the Internet environment. The REDUCE system's strategies include a novel consistency model and responsiveness techniques that permit arbitrary order in executing independent operations.

Computer-supported cooperative work (CSCW) or groupware systems allow physically dispersed teams to engage in a common task by providing an interface to a shared workspace.[1] Real-time cooperative editing systems extend the usefulness of CSCW applications by allowing team members to simultaneously view and edit shared documents. Both the CSCW and distributed computing communities have addressed the technical challenges of these systems since the mid-1980s (for example, see Ellis and Gibbs[2] and Rodden[3]). More recently, commercial products such as Microsoft NetMeeting have emerged to support some cooperative editing processes (see http://www.microsoft.com/windows/NetMeeting/Features/).

Our research investigates the principles and techniques underlying the construction of cooperative editing systems that feature

- *Real-time responsiveness*. The response to local user actions cannot exhibit noticeable delay, and the latency for remote user actions should be low. The key performance parameter here is the response time observable by the user, rather than the number of operations per second as in noninteractive application systems.
- *Distributed user community*. Cooperating users must be able to work on different machines connected over the Internet with nonnegligible and nondeterministic latency. While fiber-optic communication technologies offer virtually unlimited Internet bandwidth, the communication latency over an intercontinental connection cannot be reduced much below 100 milliseconds due to the speed limit of electronic and light signals. It is therefore communication latency, rather than bandwidth, that challenges the design of Internet-based, response-sensitive systems and calls for latency-tolerant or latency-hiding technical solutions.
- *Unconstrained operation*. Multiple users must be allowed to edit the document freely at any time to facilitate a natural cooperative infor-

1089-7801/00/$10.00 ©2000 IEEE

mation flow. Many CSCW applications have failed because they imposed too many restrictions on the users, as pointed out by Grudin over a decade ago.[4] The major challenge is to manage the multiple streams of concurrent activities so that system consistency can be maintained.

We are addressing these issues through a variety of collaborative projects known as the REDUCE system (Real-time, Distributed, Unconstrained Collaborative Editing). We have reported major theoretical and algorithmic work elsewhere.[5] In this article, we focus on our approach to achieving high responsiveness in the Internet environment in the face of high concurrency and nondeterministic communication latency. We begin by outlining the theoretical and algorithmic background for achieving high responsiveness, and then describe our REDUCE prototype and its responsiveness techniques at the system design and implementation levels. We also present performance measurement results.

REPLICATION AND CONSISTENCY MODEL

There are two different system architectures for storing shared documents: centralized and replicated. *Centralized architectures* use a single site to save the shared documents, and all updates to the shared documents are directed to this single site. Despite the advantage of simplicity, the centralized approach exhibits poor responsiveness in the Internet environment. *Replicated architectures*, on the other hand, replicate the shared documents to the local storage of each participating site, so updates can be performed at local sites immediately and then propagated to remote sites.

We adopted the replicated architecture for REDUCE to achieve better responsiveness across the Internet. Accordingly, we inherit the challenge of controlling concurrency to maintain consistency in the replicated documents. Concurrent generation of operations and nondeterministic communication latency create three major inconsistency problems:

- divergent results, caused by operations arriving and executing at different sites in different orders;
- causality violations, resulting from operations arriving and executing out of their natural cause-effect order; and
- intention violations, caused by an operation's actual effect when executed differing from its intended effect when generated.

The intention-preservation property lets users see the effects of individual operations immediately.

To address these inconsistency problems, we have proposed a consistency model (detailed in Sun et al.[5]) that has three corresponding properties:

- the *convergence* property, which ensures the consistency of the final results *at the end* of a cooperative editing session;
- the *causality-preservation* property, which ensures that dependent operations are executed in their natural causal-effect order *during* a cooperative editing session; and
- the *intention-preservation* property, which ensures that the effect of executing an operation at remote sites is the same as executing it at the local site at the time of its generation, and which also ensures that the execution effects of independent operations do not interfere with each other.

Our consistency model imposes an execution order constraint only on dependent operations, leaving independent operations open as long as the convergence and intention-preservation properties are maintained. This feature lays the theoretical foundation for achieving good responsiveness by permitting local operations to execute immediately after their generation. Moreover, the intention-preservation property lets users see the effects of individual operations immediately while protecting against interference from other operations.

The consistency model specifies what assurance a cooperative editing system promises to its users and what properties the underlying concurrency control mechanisms must support. For causality-preservation, we used the well-known technique of state-vector–based time-stamping to selectively delay some dependent operations if they arrive out of causal ordering.[2,5] To support both convergence and intention-preservation under the constraints of high responsiveness, we devised an optimistic concurrency control technique, called operational transformation.[2,6,7] The novelty of operational transformation is that it allows independent operations to be executed in any order (hence local operations can be executed immediately) but ensures that their final effects are identical and intention-preserved. We have shown elsewhere that some intention-preserved results achieved by operational transformation are not achievable by any traditional serialization protocols.[5] For an integrative review of the major issues and algorithms in operational transformation, readers are referred to Sun and Ellis.[7]

The REDUCE session manager is replicated in the prototype to enhance its reliability.

THE REDUCE PROTOTYPE

We implemented a prototype to support the REDUCE system's consistency model. Real-time responsiveness has been taken into account in the prototype architecture as well as the system design and implementation levels.

System Architecture

A REDUCE system consists of multiple cooperating *sites*. Each REDUCE site is typically a PC or a workstation, with user interface facilities for displaying shared documents and generating editing operations; local storage facilities for storing replicates of shared documents; and computing and communication facilities for executing, synchronizing, and propagating editing operations.

The REDUCE end-user–related software consists of a signed Java applet; when downloaded, it runs locally as a REDUCE site server. In addition, a centralized REDUCE session manager—a Java application running as a daemon—is used to implement protocols for cooperating membership and session management. The session manager is not involved in executing or propagating editing operations; it is contacted only when a user joins or leaves a session. Because communication among sites for multicasting editing operations is direct, rather than via the session manager, network latency is reduced to the minimum. In addition, the session manager is replicated in the prototype to enhance its reliability so that it can tolerate single-site (session manager) failure.

Fault-Tolerant Session Manager

Figure 1 depicts a REDUCE system consisting of a replicated session manager (SM) and three editing sites. The SM has a session manager connector (SMC) component waiting for connection requests from the REDUCE Java applets via the (editing) site server. Once a connection request is accepted, the SMC creates a new thread, called the site handler (SH), to handle further communications between the editing site and the session manager.

For each cooperative editing session, the session manager maintains a session table with one entry for each potential editing site. When a new connection request arrives to join an existing session, the session manager searches the corresponding session table for a free entry, allocates the free entry's index to the new site as its session identifier, and creates a site handler to handle further communications with the new site. When an editing site leaves a session, the corresponding session table entry is reclaimed.

The session manager plays a central role in looking after the joins and leaves of team members. With a single session manager server, no new team member can join the cooperative editing session, and no proper notification can be propagated to other members regarding the leave of a team member if the server is down. To increase REDUCE reliability in the Internet environment, we have used a variation of the primary-backup model (see Budhiraja et al.[8]) to tolerate the single server failure as depicted by the backup session manager in Figure 1. The twin session manager servers are located at different sites over the Internet and connected to each other via twin listener and connector threads. If the primary session manager server is down, the backup session manager site will automatically replace the primary site to manage the editing sessions, and vice versa.

Site Server

The editing site server is implemented as a Java applet that can be downloaded by any team member joining an editing session, as shown in Figure 1. When the applet is downloaded, the site server will contact the session managers—the twin servers for primary and backup SMs—to make a connection with other team members in the same session and thereafter will be notified by the active session manager when any team member joins or leaves the ses-

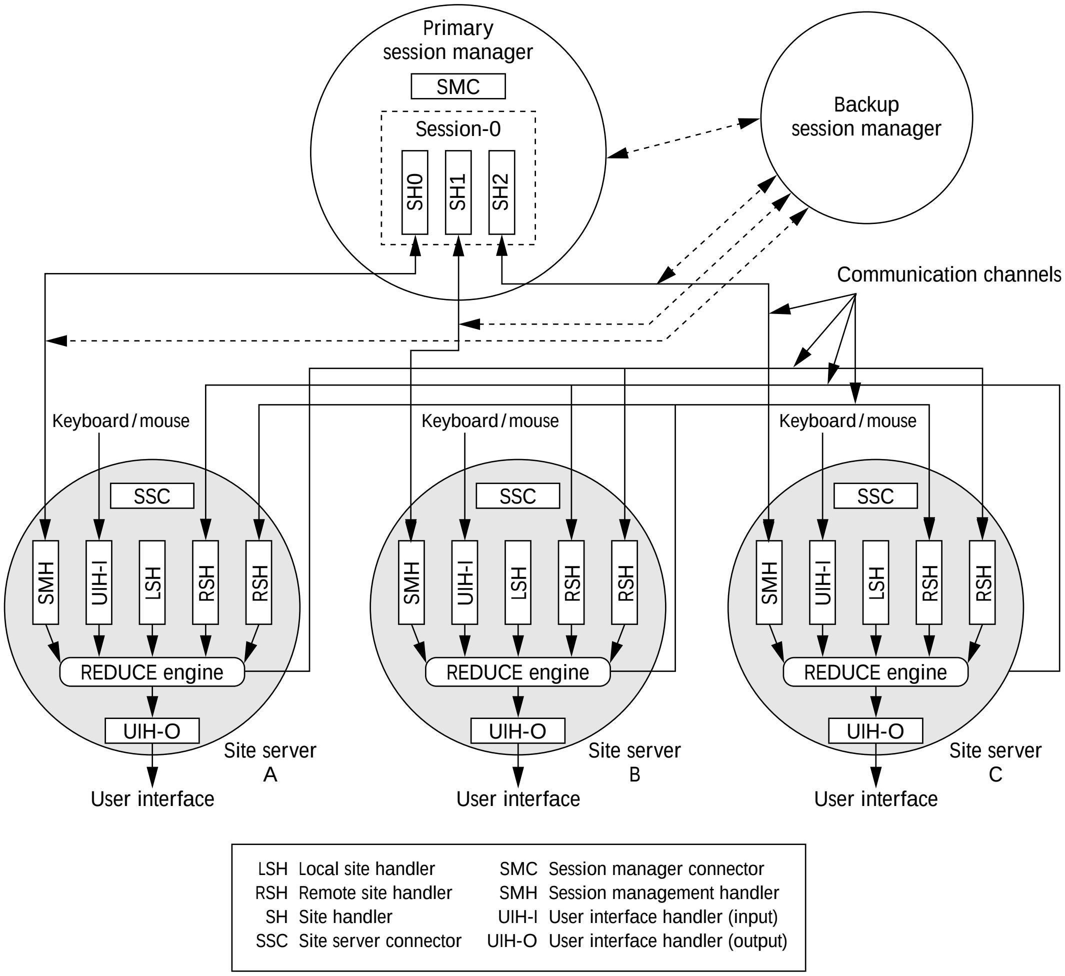
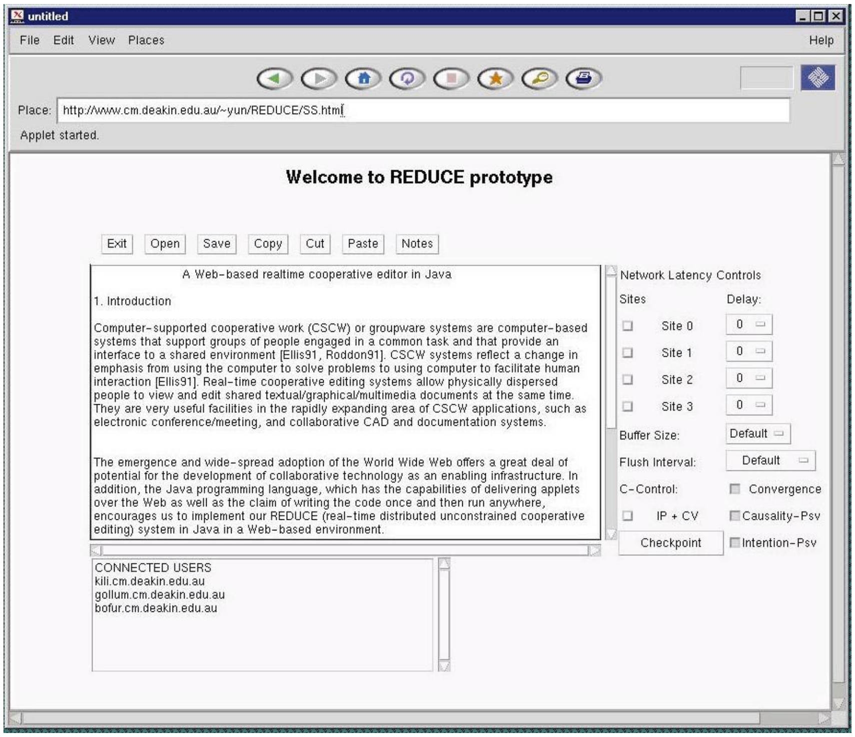


Figure 1. REDUCE architecture and components. Primary-backup servers provide fault-tolerant session management, and site server components enable dynamic cooperative editing sessions.

sion. The site server consists of the following major components:

- A session management handler (SMH), which is a thread inside the site server process that handles all messages related to the session manager.
- A site server connector (SSC), which is a thread responsible for accepting connection requests from new editing sites and for creating a new remote site handler (see the next item) for each new connection.
- Multiple remote site handlers (RSH), which are concurrent threads corresponding to multiple remote sites in the current cooperative session. Each remote site handler is responsible for receiving messages from the corresponding remote site. An RSH may be created by the SMH thread if the local site is newly joined, or by the SSC thread if the local site is an existing site.
- A local site handler (LSH), which is a thread responsible for handling some miscellaneous issues, such as the local timers for flushing the input string buffer and for multicasting local status messages to remote sites to facilitate garbage collection if the local user has been silent for a certain period of time.
- A user interface handler (UIH), which runs inside the main thread of the site server process and implements the local graphics interface. This component is conceptually divided into two parts: one is the UIH Input (UIH-I) for handling input events generated from the local keyboard and mouse; the other is the UIH Output (UIH-O) for handling requests (from the REDUCE engine (see the next item) for



Place: http://www.cm.deakin.edu.au/~yun/REDUCE/SS.html

Applet started.

Welcome to REDUCE prototype

A Web-based realtime cooperative editor in Java

1. Introduction

Computer-supported cooperative work (CSCW) or groupware systems are computer-based systems that support groups of people engaged in a common task and that provide an interface to a shared environment [Ellis91, Roddon91]. CSCW systems reflect a change in emphasis from using the computer to solve problems to using computer to facilitate human interaction [Ellis91]. Real-time cooperative editing systems allow physically dispersed people to view and edit shared textual/graphical/multimedia documents at the same time. They are very useful facilities in the rapidly expanding area of CSCW applications, such as electronic conference/meeting, and collaborative CAD and documentation systems.

The emergence and wide-spread adoption of the World Wide Web offers a great deal of potential for the development of collaborative technology as an enabling infrastructure. In addition, the Java programming language, which has the capabilities of delivering applets over the Web as well as the claim of writing the code once and then run anywhere, encourages us to implement our REDUCE (real-time distributed unconstrained cooperative editing) system in Java in a Web-based environment.

CONNECTED USERS
kili.cm.deakin.edu.au
gollum.cm.deakin.edu.au
bofur.cm.deakin.edu.au

Network Latency Controls

Sites	Delay:
☐ Site 0	0
☐ Site 1	0
☐ Site 2	0
☐ Site 3	0

Buffer Size: Default
Flush Interval: Default
C-Control: ☐ Convergence
☐ IP + CV ☐ Causality-Psv
☐ Intention-Psv

Figure 2. User interface to the REDUCE prototype.

updating the local editing window (that is, the shared document state).

- A REDUCE engine (RE), which is the central component shared by all other components inside the site server process. It realizes the REDUCE consistency model, takes care of communication with remote site servers, and provides a synchronized point of access to the local UIH-O and to other local site status information by multiple concurrent threads (that is, RSHs, UIH-I, LSH, and SMH) inside the site server process. The RE is implemented as a monitor, so there can be only one active thread inside it at any instant of time.

Implementation-Level Responsiveness Strategies

In the context of text editing, good responsiveness means that the effect of a so-called *character-wise operation*—that is, hitting a key to insert or delete a character—is reflected immediately on the local interface. Propagating each character-wise operation to remote sites over the Internet is not communication-efficient, so REDUCE uses an automatic character-to-string conversion scheme. When a character-wise operation is generated, it is executed immediately and saved in an internal buffer at the local site; the accumulated character-wise operations will be converted into a so-called *string-wise operation* under the following circumstances:

- A newly generated character-wise operation cannot be combined with the accumulated operations as a single string-wise operation. For example, if after executing a sequence of consecutive character-wise insertions the user moves the cursor to a nonconsecutive position or starts issuing a different type of operation (such as deletion), the accumulated operations should be propagated, since they cannot be merged with forthcoming operations into a single string-wise operation.
- A newly generated character-wise operation cannot share a common state-vector time stamp with the accumulated operations.[5] After processing a remote operation, for example, the accumulated local operations must be propagated, since future local operations will carry different time stamps.
- The number of accumulated operations has reached the system/user-selected space/time limits. This happens when, for instance, the buffer is about to overflow or when a timer expires after the user interface has been idle for a certain period of time.

- The user has initiated a synchronization/checkpoint operation from the interface to flush accumulations in the local buffer to remote sites.

The thread that handles local operations has been assigned a higher priority than threads for handling remote operations, so that whenever a local operation has been generated, the local thread is guaranteed to seize the REDUCE engine for executing this operation as soon as the current thread (if any) inside the REDUCE engine finishes. With this priority scheme, the local thread's longest possible waiting time is bound to the time for completing a single remote operation, regardless of how many remote operations have been waiting.

User Interface

The screen snapshot in Figure 2 depicts a REDUCE text editor launched from the Sun HotJava Web browser:

- The text-editing panel allows the team member to edit the document from any position at any time.
- Buttons for file access and editing operations (copy, cut, paste, and so on) let team members prepare text in a private window before putting it in the shared editing panel.
- A display panel indicates the cooperating sites in the current session.
- Control buttons (on the right in Figure 2) allow users to select different buffer sizes for saving character-wise operations and different time intervals for automatically converting character-wise operations into string-wise ones and flushing them to remote sites. These buttons are normally used only by REDUCE developers for experimental purposes, such as controlling network delay parameters for different sites, selecting various combinations of consistency properties, and so forth.

PERFORMANCE EVALUATION

Research has shown that users do not notice delays when response times are less than 0.10 seconds in interactive user interfaces.[9] We conducted performance tests on the REDUCE prototype, which showed its response times to be well below this threshold of noticeable delay.

Our performance tests were conducted on a Sun workstation running Solaris 2.6 with a 100-MHz CPU. The time measurement is accurate to milliseconds. Each specific time interval is measured by recording and subtracting two CPU times immediately before and after the related task or task sequence. For example, the CPU time consumption for processing a local insert is the time interval starting with the key event and ending when the character displays on the screen.

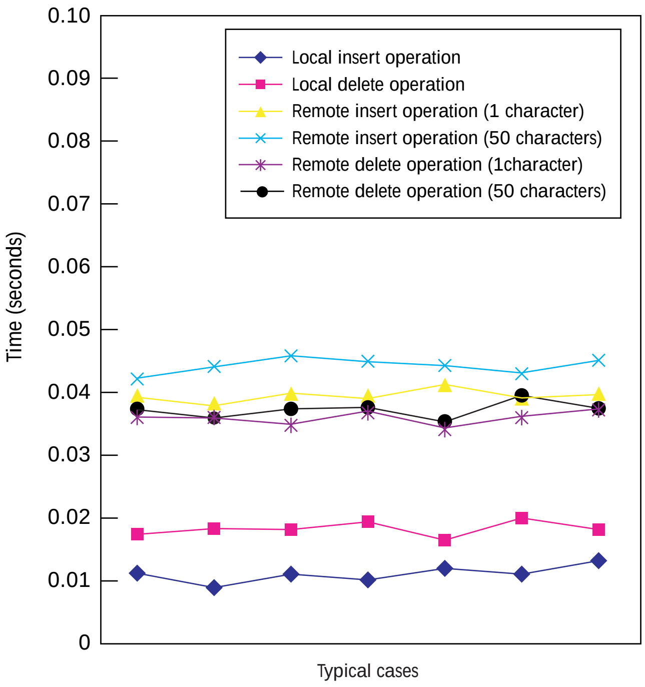


Figure 3. REDUCE prototype performance on some typical cases without interference. In processing individual local and remote edit operations, the REDUCE prototype performed well below the threshold of user-noticeable delay, which is 0.10 seconds.

As shown in Figure 3, on average, the performance curves for local single-character insert and delete operations are far below the user noticeable delay mark of 0.1 seconds. This is true even if characters are typed at a very fast pace. Note that the local deletion time is a bit longer than the local insertion time. This is because a REDUCE delete operation must collect and save the deleted text for future operational transformation against remote operations, while an insert operation has already had the inserted character when the key is pressed.

We also measured the execution time of remote string-wise operations to evaluate the benefits of propagating and executing them. In Figure 3, the performance for remote operations processing (including the time of operational transformation and execution) is depicted for string lengths of one character and 50 characters. As shown, the time for

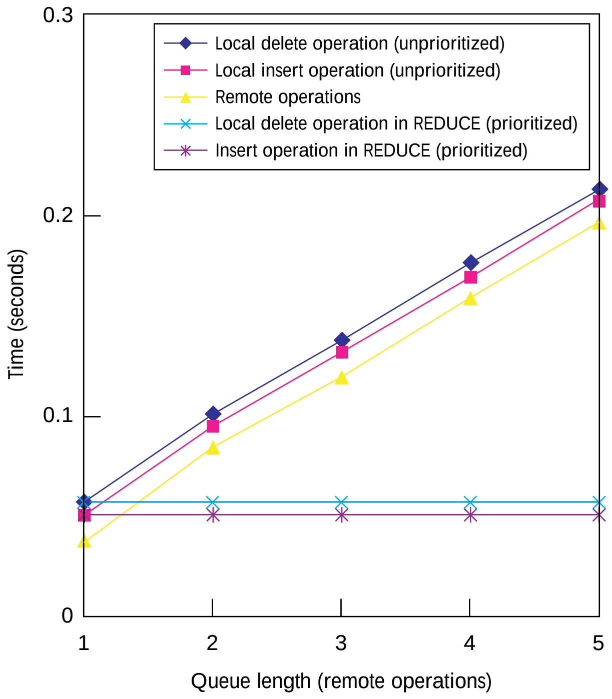


Figure 4. Performance comparison of local edit operation processing. Without prioritization, the delay of a local operation becomes noticeable to users whenever remote operations are waiting in the queue.

processing a remote operation (either a one-character or a 50-character string) is very short (less than 50 ms), and the difference between processing a one-character string and a 50-character string is very small. If the propagation strategy for the string-wise operation was not used, a 50-character string would cause 50 multicasting messages over the Internet and 50 one-character operation transformations and executions, which would be nearly 50 times slower than processing a single 50-character string-wise operation. In other words, the string-wise strategy is very effective in reducing the number of communication messages, transformations, and executions.

Based on the measurement of processing times for both local and remote operations, it is straightforward to evaluate the effectiveness of the scheme for setting a higher priority to the local operation-handling thread in REDUCE. Without such a priority strategy, the user would easily notice the delay of a local operation whenever remote operations were waiting in the queue, as illustrated in Figure 4. Under these circumstances, the response time for a local operation would be proportional to the number of remote operations waiting in the queue, and would cross the user-noticeable delay line when up to two remote operations were waiting.

With a higher priority scheme in REDUCE, however, a local edit operation waits, in the worst case, only for the processing of one ongoing remote operation, regardless of how many remote operations are waiting in the queue. This strategy results in a flat worst-case performance curve as shown in Figure 4, which is well below the user-noticeable delay line. In other words, it is essential to grant a higher priority for local operations, particularly when a large number of remote operations have accumulated in the queue.

In reality, the response time of a local operation depends on the current load of the workstation as well as the CPU time consumed inside the REDUCE prototype. However, computers are normally operated by single users with relatively light loads. When we measured the CPU time, we also measured the real-world time for editing operations. It is clear that only a fraction of extra time is added to the CPU time as the real response time. Therefore, the conclusions drawn from the performance results shown in Figures 3 and 4 remain valid in real-world situations.

CONCLUSIONS AND FUTURE WORK

The prototype performance evaluations indicate the effectiveness of the REDUCE strategies for delivering real-time response to users in the face of high concurrency and the nondeterministic communication delays characteristic of the Internet. Because the prototype is implemented in Java, it can run on any platform that supports the standard Java 1.1 virtual machine.

Our work on REDUCE continues in several directions among multiple collaboration teams. Ongoing research includes formalization and verification of the consistency model and concurrency-control algorithms as well as development of support for group awareness in cooperative editing environments. We are investigating the application of REDUCE collaborative technologies and systems to graphics editing, distance education, and collaborative programming (for links to different projects and prototypes, see http://www.cit.gu.edu.au/~scz/reduce).

Under another umbrella, we are working on a process-centered, Web-based, teamwork environment that integrates REDUCE as a brainstorming and co-design tool. ■

ACKNOWLEDGMENT

The work reported in this article has been supported by grants from the Australian Research Council (No. A49601841 and No. 000000711), Hong Kong CERG (No. 9040442), and some internal grants from Deakin University, Griffith University, University of Southern Queensland, and City University of Hong Kong. Yang's work in this project was carried out mainly when he was with Deakin University. We are grateful to David Temple, in particular, for his assistance in implementing the backup session manager server and for conducting the performance measurement. We also wish to thank the anonymous referees for their valuable comments and suggestions that helped improve the final presentation of this article.

REFERENCES

1. C.A. Ellis, S.J. Gibbs, and G.L. Rein, "Groupware: Some Issues and Experiences," *Comm. ACM*, vol. 34, no. 1, Jan. 1991, pp. 39-58.
2. C.A. Ellis and S.J. Gibbs, "Concurrency Control in Groupware Systems," *Proc. ACM SIGMOD Conf. Management of Data*, ACM Press, New York, 1989, pp. 399-407.
3. T. Rodden, "A Survey of CSCW Systems," *Interacting with Computers*, vol. 3, no. 3, Dec. 1991, pp. 319-353.
4. J. Grudin, "Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces," *Proc. ACM Conf. CSCW*, ACM Press, New York, 1988, pp. 85-93.
5. C. Sun et al., "Achieving Convergence, Causality-Preservation, and Intention-Preservation in Real-Time Cooperative Editing Systems," *ACM Trans. Computer-Human Interaction*, vol. 5, no. 1, Mar. 1998, pp. 63-108.
6. M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhauser, "An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors," *Proc. ACM Conf. CSCW*, ACM Press, New York, 1996, pp. 288-297.
7. C. Sun and C. Ellis, "Operational Transformation in Group Editors: Issues, Algorithms, and Achievements," *Proc. ACM Conf. CSCW,* ACM Press, New York, 1998, pp. 59-68.
8. N. Budhiraja et al., "The Primary-Backup Approach," in *Distributed Systems*, 2nd ed., Addison-Wesley, Reading, Mass., 1993, pp. 199-216.
9. B. Broom, *Aspects of Interactive Program Display*, doctoral dissertation, Dept. of Computer Science, Univ. of Queensland, Australia, 1987.

Yun Yang is currently an associate professor in the School of Information Technology, Swinburne University of Technology, Melbourne, Australia. He received a PhD in computer science from the University of Queensland, Australia, and an ME in computer science from the University of Science and Technology of China. His research interests include Internet and Web-based computing technologies and applications, real-time CSCW and groupware systems, visualized (agent-based) workflow environments for software development and e-commerce, and mobile computing systems.

Chengzheng Sun is currently a professor in the School of Computing and Information Technology, Griffith University, Brisbane, Australia. He received a PhD in computer science from University of Amsterdam, The Netherlands, and in engineering from Changsha Institute of Technology, China. His research interests include Internet and Web-based computing technologies and applications, real-time CSCW and groupware systems, distributed operating systems, mobile computing systems, and parallel logic and object-oriented programming systems.

Yanchun Zhang is currently a senior lecturer in computer science in the Department of Mathematics and Computing at the University of Southern Queensland, Australia. He has a PhD in computer science from the University of Queensland. His research interests include database design, object-oriented and multimedia databases, distributed and multidatabases, CSCW, database support for cooperative work, distributed and parallel processing, Web data management, and visual database processing.

Xiaohua Jia is currently an associate professor in the computer science department at City University of Hong Kong. He received an ScD in information science from the University of Tokyo, Japan. Jia's research interests include operating systems, distributed systems, network systems, computer communications, and real-time communications.

Readers may contact Yang at yun@it.swin.edu.au or Sun at scz@cit.gu.edu.au.