Scalability of Web-Based Electronic Commerce Systems

Gregor v. Bochmann, University of Ottawa; Johnny W. Wong and David Evans, University of Waterloo; Terence C. Lau and Don Bourne, IBM Canada; Brigitte Kerhervé, Université du Québec à Montréal; Mohamed-Vall M. Salem and Haiwei Ye, University of Montreal

Abstract

In a Web-based electronic commerce system, users browse product information offered by an online store and submit requests to purchase selected items. From the user's perspective, response time is a factor that could impact the acceptability of electronic commerce. This article is concerned with system architectures for online stores. Emphasis is placed on techniques to improve a system's capacity to support more users without suffering a noticeable degradation in response time performance. Such techniques have been investigated as part of a Canadian Institute for Telecommunications Research major project entitled Enabling Technology for Electronic Commerce Applications, developed in close collaboration with the IBM® Centre for Advanced Studies. The basic architecture of a Web-based electronic commerce system is first described along with the types of data that need to be managed. Next, an overview of existing techniques for improving system capacity is presented. Finally, highlights of research results obtained as part of the CITR electronic commerce major project are discussed.

Introduction

In the last decade, there has been growing interest in implementing electronic commerce applications on the Internet. A popular class of applications is the sale of goods and services using Web technology. For such applications, users browse product information offered by an online store. They may subsequently submit a request to purchase selected items. From the user's perspective, factors that could impact the acceptability of electronic commerce include security and trust, ease of use, and response time.

This article is concerned with system architectures supporting online stores. Since the early stages of Web development, it has been recognized that the server architecture should be scalable. By scalable, we mean a system's capacity can be improved in a straightforward manner to support more users without suffering noticeable degradation in response time performance [1]. Scalability is an important issue because the user base of online stores is large[1] and is expected to increase in the future.

Performance and scalability of Web-based electronic commerce systems have been investigated as part of the Canadian Institute for Telecommunications Research (CITR) major project entitled Enabling Technology for Electronic Commerce Applications. This major project was developed in close collaboration with the IBM® Centre for Advanced Studies (CAS) [2]. It represents an example of a very successful approach to cooperative research between industry and academia. Participants included researchers from six universities, CAS, and the Electronic Commerce Development Team at the IBM Toronto Laboratory.

For a Web-based electronic commerce system, techniques that can be used to improve system capacity include mirror sites, caching, one or more server clusters, and parallel database systems. The last two techniques are of particular interest to the CITR major project because of their immediate relevance to industry products developed at the IBM Toronto Laboratory.

In this article we first describe the basic system architecture and the types of data that need to be managed. This is followed by an overview of techniques that can be used to improve system capacity. We next describe highlights of new research results in system performance and scalability. Finally, we provide some concluding remarks.

Basic System Architecture

A typical Web-based electronic commerce system has a three-tier architecture: the Web server, electronic commerce application server, and database system (Fig. 1). The Web server is a process that handles requests from users and returns the requested Web pages. The application server contains the business logic and accesses the database for information such as

[1] *For example, Amazon.com, in its letter to shareholders, mentioned that it served 25 million customer accounts in 2001.*

catalogs, inventory level, and user information, such as registration data and shopping cart content. The three components may reside on the same machine or on different machines.

Typically, users are engaged in browsing or purchasing activities. The Web pages the system delivers to users are constructed by the business logic at the application server. This involves retrieving appropriate data items from the database, performing the required processing, and formatting the results so that the user may view them. The system also maintains a shopping cart for each user. During a shopping session, the user may add items to (or remove items from) the shopping cart. When the user is ready to submit an order, the shopping cart content will constitute the items to be purchased.

In general, the database for an online store may contain the following types of data:

- Catalog data — These are semi-static data that do not change in response to user requests. Such data are updated infrequently and only via well defined management commands.
- User data — These are dynamic data that describe user-specific information such as registration data, order history, and shopping cart content. They are created and updated in response to user requests.
- Inventory level — These data describe inventory information. They are updated in response to user requests and inventory management.

In designing system architectures for electronic commerce, an important consideration is how the various types of data are managed.

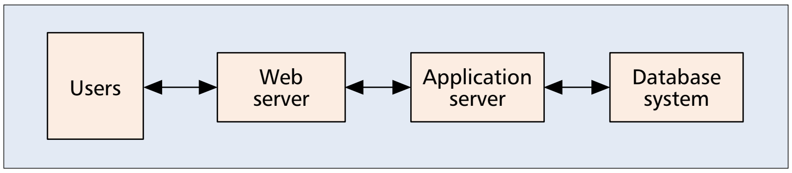

Figure 1. *Basic Web server architecture.*

Scaling Techniques

In this section we discuss techniques that can be used to increase the capacity of a Web-based system to support more users.

Mirror Sites

A popular method to improve a system's overall capacity is to use mirror sites. With this method, information requested by users is made available at multiple server sites; each site has its own copy of the database. This is essentially an extension of the architecture shown in Fig. 1 where the system is replicated, with one replica per site. Each replica has a different URL. Typically, a user selects a server site from a list of sites that offer the same service. The selection is often made according to geographical proximity or the user's perception of the load at the various sites. The overall capacity is increased because each site can process requests independent of the others. However, consistency of the databases at the various sites must be maintained, and there is no formal method to do this. This issue, in the context of electronic commerce applications, will be addressed later in this article when multiple cluster architecture is discussed.

Caching

Another popular approach to improve a system's overall capacity is caching. Web pages may be cached at the user's machine. Subsequent requests for these pages can be serviced by the copies in the cache without the need to access the Web server. System capacity is effectively improved because fewer requests are submitted to the system. Caching may also be done at the application server. In this case, Web pages are stored at the application server after their construction. If the same page is required to service two or more user requests, only the first request will incur costs in executing the business logic and accessing the database.

Furthermore, Web pages may be cached at proxy servers that are located between the user and the Web server [3]. For a given user, requests are always sent to the proxy with whom the user is associated. If the requested page is available at the proxy, the proxy can process the request directly; otherwise, the request is forwarded to the Web server. In general, proxies can be organized hierarchically to further enhance the scalability of the overall system.

Note that in electronic commerce applications, caching is most easily applied to browsing of information pages. Requests to change the shopping cart content or transactions are usually sent to the Web server directly.

Server Cluster

A server cluster consists of a number of *server nodes* running on multiple machines. Each server node processes user requests independent of the others. As the demand grows, the system capacity can be increased by deploying more server nodes. Unlike mirror sites, the server nodes in a cluster are transparent to users in the sense that a single public URL is used to access the cluster. With respect to response time performance, an important issue is load balancing among the server nodes. Examples of load balancing techniques are domain name system (DNS)-based and dispatcher-based load balancing.

When a user accesses a URL, a request is first sent to a name server, where the host name in the URL is mapped to an IP address. This is the IP address to which the request is sent. In the DNS-based approach, the name server is set up so that the IP addresses of the various server nodes are selected according to a load-balancing algorithm. A common algorithm is round-robin (RR) where server nodes are selected in cyclic order (i.e., node 1, node 2, …, node N repeatedly), where N is the number of server nodes. This approach was used in an early version of the National Center for Supercomputing Applications Web server.

In dispatcher-based load balancing, a dispatching entity is placed between the users and the server cluster. This dispatcher is typically configured with a public IP address that represents the cluster. All packets sent to the cluster are first routed to the dispatcher, which distributes the packets to the server nodes. The communication between the dispatcher and server nodes is hidden from the users. Examples of

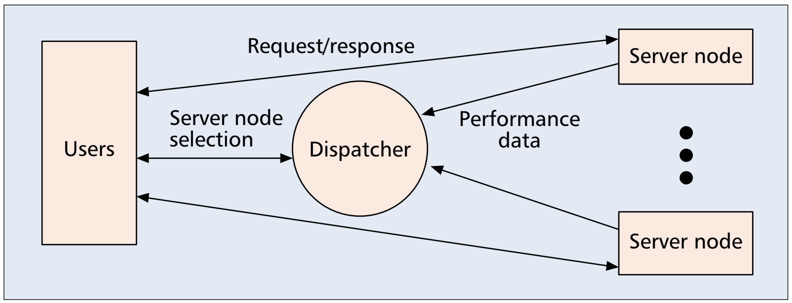

Figure 2. *Dispatcher-based load balancing.*

dispatcher-based load balancing include ONE-IP developed at Lucent Technologies, IBM eNetwork Dispatcher, and Cisco LocalDirector.

In ONE-IP, the dispatcher distributes TCP connections among server nodes. Packets belonging to the same connection are forwarded to the same node. For a given TCP connection, the selection of a server node may be based on a hash of the user address. Alternatively, each server node may handle requests from a fixed and disjoint portion of the user address space. IBM eNetwork Dispatcher uses a TCP connection forwarding mechanism. It runs on the same LAN as the server nodes. These nodes and the dispatcher share the same IP address. The system is configured such that the dispatcher receives all incoming packets sent to that IP address. The dispatcher processes these packets, selects a server node for each new connection, and forwards all packets related to that connection to the selected server node. Outgoing data are sent directly from server node to user without any involvement of the dispatcher. In contrast, Cisco LocalDirector is set up to intercept all packets belonging to each TCP connection. It performs the necessary changes in IP packet headers so that the mapping between the user and the selected server node remains transparent to both.

Parallel Database Systems

The capacity of an individual server cluster can be enhanced by using a parallel database system. Typically, a parallel database system consists of several database nodes; each is capable of processing queries independent of the others. The amount of parallelism is affected by how the data are distributed across the database nodes. For example, if frequently requested data are replicated at two or more nodes, system capacity is increased because two or more queries may be processed simultaneously.

Existing data distribution strategies developed for parallel database systems usually try to minimize the traffic between database nodes or to balance the load among these nodes in terms of total size of data or disk access frequency [4]. In general, determining the optimal distribution of data across the nodes of a parallel database is a difficult problem, and heuristic solutions are often developed. These solutions generally lead to improvements in overall system capacity.

Research Results

In this section we present highlights of research results obtained as part of the CITR major project on electronic commerce.

Performance of Load Balancing Algorithms

Within a server cluster, poor server node selection may lead to some nodes being saturated while other nodes have surplus capacity. Server node selection to achieve load balancing is therefore an important issue [5]. We have investigated the performance of dispatcher-based load balancing algorithms. A salient feature of our system architecture is *session-based*[2] server node selection in which the dispatcher assigns a user to a server node for an interval of time called the *assignment period* (Fig. 2). This is different from the above-mentioned approaches where the dispatcher is involved on a per-TCP connection basis. In our architecture the dispatcher is only involved in the initial server node selection, and the user interacts directly with the selected node during the assignment period. This would tend to reduce the dispatcher load. Furthermore, performance data can be collected at the server nodes and sent to the dispatcher. These data may be used for load balancing purposes.

We have evaluated the performance of the following load balancing algorithms using simulation:

- Weighted RR (WRR) — Server nodes are selected in cyclic order with some nodes being selected more frequently than others. Note that RR is a special case of WRR.
- Least Utilization (LU) — The server node with the lowest utilization is selected; the details of this algorithm can be found in [6]. Note that server node performance data are used in LU, but not in WRR.

The details of our simulation model, including the workload parameters, are described in [6]. A performance measure of interest is the response time percentile $T(x)$ = Prob [response time $\leq x$]. The percentile is considered instead of the mean because it is a better indicator of good user experience. Our results show that for homogeneous server nodes (i.e., each node in the cluster has the same capacity), a simple algorithm such as RR is effective with respect to load balancing.

For heterogeneous server nodes (i.e., some nodes are fast while others are slow), the results for $T(x)$ are shown in Fig. 3. We observe that RR yields significant imbalance in response time performance among the server nodes. This is because RR does not distinguish between nodes of different capacities, resulting in the fast nodes being underutilized while the slow ones are overloaded. Imbalance is not observed when either WRR (with weights determined according to server node capacity) or LU is used. Between them, WRR has the advantage of simplicity, assuming that the weights can be determined efficiently. It should be the preferred algorithm unless the available capacities at the various server nodes change frequently. LU is more capable of adapting to changes in available capacities, and would be a good alternative in such dynamic environments.

Multiple Cluster Architecture

We have designed a highly scalable architecture for electronic commerce systems using multiple clusters [7]. Each cluster has multiple server nodes and a single database. As demand grows,

[2] *In our context, a session is a time interval during which the user interacts directly with a selected server without the involvement of the dispatcher.*

system capacity can be increased by deploying more clusters. In addition to load sharing among the clusters, users can be redirected to another cluster in case of failure or scheduled maintenance, thus improving the availability of the overall system.

A salient feature of our architecture is the specific way that different types of data are managed. Since each cluster has its own database, any updates to the catalogs must be made to all the databases. Also, if a user currently interacting with cluster 1 were to be redirected to cluster 2, the shopping cart content for this user must be available at cluster 2 before the user can view the existing items or add more items. Consistency of the databases at the various clusters needs to be maintained. This incurs overhead, and some of the data may temporarily be out of date. This is the price to pay for increased scalability and enhanced availability.

Our architecture is depicted in Fig. 4. For ease of exposition, only two clusters are shown: *primary* and *secondary*. In general, there may be more than two clusters in the system. For reliability reasons, several clusters may assume the role of the primary cluster; this allows a new primary to be chosen in case the current primary fails. The system maintains, for each user, a set of clusters to which the user may be assigned (referred to as the user's *cluster set*). Clusters in this set may be determined by a combination of factors such as user/cluster affinity, cluster availability, and global load balancing considerations. When a user logs on to the system, the load balancer assigns the user to a cluster within his/her cluster set. This cluster is referred to as the user's *home cluster*. The user will then interact with this cluster. If the user is inactive for an extended period time (e.g., an hour), he/she may be migrated to another cluster within the cluster set (this becomes the user's new home cluster). Such an action is usually taken to achieve global load balancing. Also, when a cluster fails or is taken out of service to perform hardware or software maintenance, all users interacting with that cluster are redirected to new home clusters selected from their respective cluster sets.

We next describe how the different types of data are managed. The primary cluster is the central repository of the catalogs. Any updates to catalogs are made at this cluster and then pushed to the secondary clusters. The primary is also responsible for inventory management. Specifically, a portion of the available inventory is assigned to each cluster. The assignment may be based on the expected demands at the various clusters. When the available inventory has dropped below a given threshold, or if there is insufficient inventory at a cluster to process an order, the primary cluster may be contacted to request reallocation of inventory.

Dynamic data such as shopping cart content are created and updated at the user's home cluster. Periodically, this home cluster transmits updated information to the primary cluster, which then distributes it to the other clusters in the user's cluster set. The shopping cart content will then be available if a user is redirected to another cluster. Finally, when a user places an order, the shopping cart content is converted into a transaction that is forwarded to the primary cluster for processing.

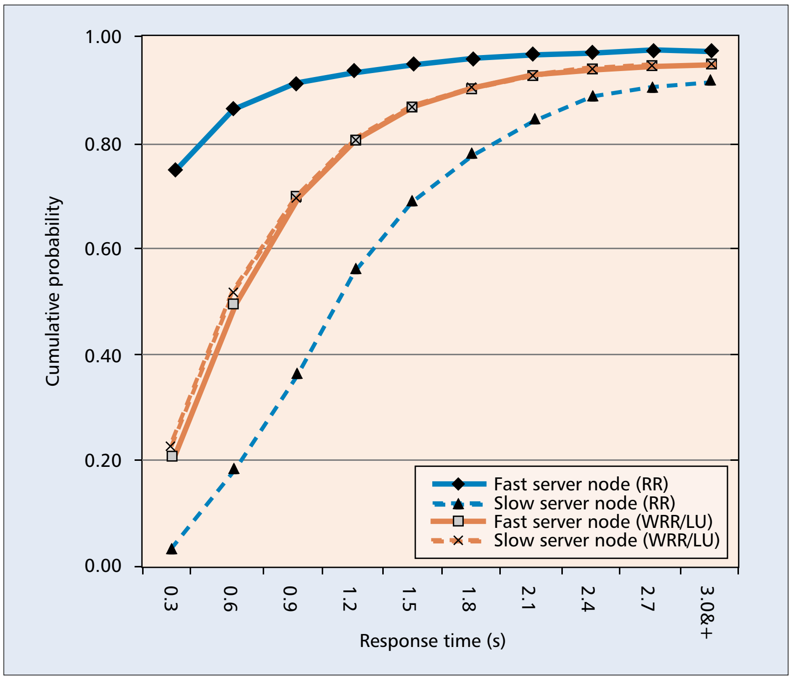

Figure 3. *Response time performance for heterogeneous server nodes.*

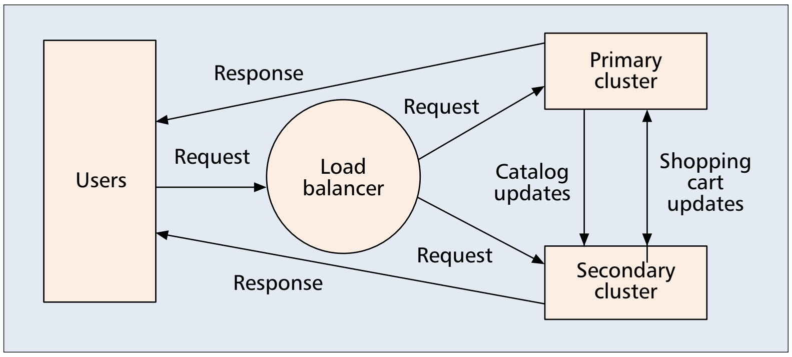

Figure 4. *Multiple cluster architecture.*

Our system architecture has been tested with IBM WebSphere® Commerce and IBM DB2®. Our implementation, together with measurement results, has confirmed the feasibility of our multiple cluster design.

Staleness/Resource Consumption Trade-off

In our multiple cluster architecture, a data item at the secondary cluster becomes stale (or outdated) when the same data item is updated at the primary. Staleness can be eliminated if the primary cluster notifies the secondary cluster immediately upon completion of each update. This technique can be resource-intensive if the data are updated frequently. In some cases, staleness is unavoidable because the updates may occur at a rate faster than the primary can transmit them, or the primary may not wish to expend the processing and communication resources required to transmit all updates. There is therefore a trade-off between resource consumption and staleness. We are the first to characterize this trade-off using analytic modeling.

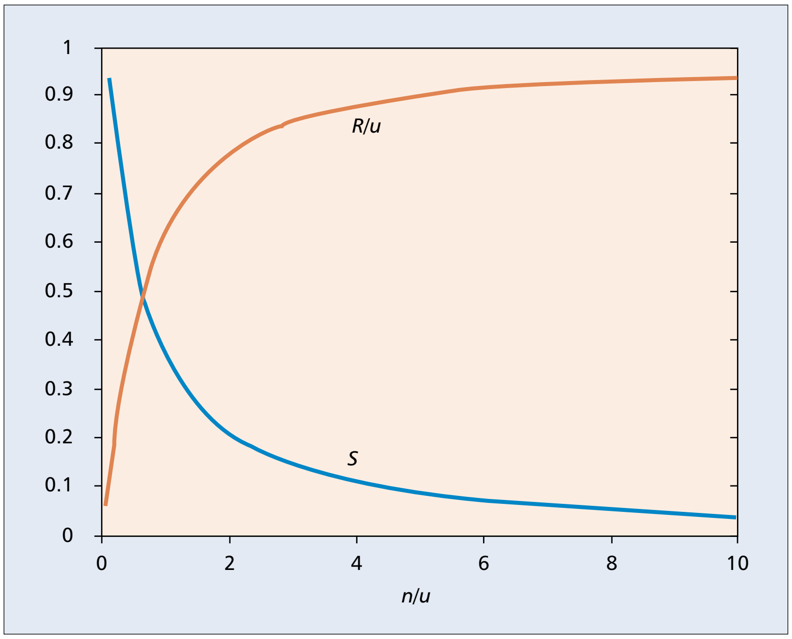

Figure 5. *Staleness/resource consumption trade-off.*

In our investigation, a measure of staleness was defined [8]. Suppose a given data item is updated at the primary cluster at time t_1. Staleness is not incurred if this update is transmitted immediately to the secondary cluster. On the other hand, if the update is transmitted some time later, say at t_2, the data item is considered to be stale from t_1 to t_2. The staleness of a data item is defined to be the fraction of time the data item is stale. Our definition of staleness is different from others [9] in the sense that the primary cluster is viewed as being responsible for keeping the data item current. Any staleness is caused by the primary cluster not being able to transmit updates promptly.

We also defined a *transmission attempt* at the primary. Specifically, a transmission attempt for a given data item involves the following activity. The status of the data item is checked; if an update has been made since the last attempt, the update is transmitted to the secondary. Our analytic result shows that over the long term, making transmission attempts periodically is optimal as far as minimizing staleness is concerned [8]. Minimum staleness is given by

$$S = \frac{e^{-u/n} + u/n - 1}{u/n}, \qquad (1)$$

where u is the update rate and n is the rate at which transmission attempts are made. Analytic results were also obtained for R, the rate at which processing resources are consumed in order to transmit page updates [8]. It was found that R is related to the minimum staleness S by

$$R/u + S = 1. \qquad (2)$$

The trade-off between staleness and resource consumption is shown in Fig. 5. In this figure, n/u and R/u are rates relative to the update rate. We observe that when n/u is small, an increase in n results in a significant improvement in staleness, but at the expense of a considerable increase in R. The amount of improvement (and the corresponding increase in R) is a decreasing function of n.

Parallel Database Systems

In our CITR major project, the issue of data distribution in a parallel database system has been investigated. We distinguish between data that may be replicated at two or more database nodes and data that are stored at one database node only. In electronic commerce applications, catalog data are frequently requested, whereas their update rate is low. They are natural candidates for replication because the same data will likely be required by two or more users simultaneously, and queries involving such data may be processed in parallel at the various nodes. On the other hand, dynamic data such as shopping cart content are frequently changed, and are usually required by only one user. Replication is not desirable because of the small chance of parallel execution of queries about the same data, and because overhead is incurred to keep the replicated data consistent. Shopping cart content is therefore a good candidate for partitioning.

We have developed a heuristic solution to the problem of distributing data optimally across database nodes. For convenience, we will describe our heuristic using a relational database system. The first step of our heuristic is to assign each individual table in the database to one of two sets: S_r for replication or S_p for partitioning [10]. This assignment is based on the frequency at which updates to the tables are made. Such information can be obtained by examining the executed SQL statements. Tables with update frequency above (or below) a given threshold are placed in S_p (or S_r).

Tables belonging to S_r are replicated to two or more database nodes. For tables in S_p, our strategy is to provide better performance for the more frequently invoked queries. When processing a query, access to two tables may be required, say, in a join operation. To reduce the amount of traffic between database nodes (and thereby reduce the mean query execution time), rows that are frequently used in the same operation should be located at the same node. In our heuristic, statistics on such frequencies are collected. The tables in S_p are then partitioned and stored such that internode communication is minimized.

Our data distribution strategy has been tested with IBM DB2 Universal Database Extended Enterprise Edition.[3] Our implementation, together with measurement results, has confirmed the feasibility of our heuristic method [10].

Concluding Remarks

We have reviewed various system issues in improving the performance and scalability of Web-based electronic commerce systems. Our review has included new research results in load balancing algorithms for a server cluster, the design of a multiple cluster architecture, the staleness/resource consumption trade-off in maintaining consistency among database copies, and the use of a parallel database system for capacity enhancement. These results are highly relevant to the understanding of performance and scalability of electronic commerce systems.

[3] *IBM, DB2, DB2 Universal Database and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries or both. Other company, product and service names may be trademarks or service marks of others.*

The techniques mentioned in this article are based on exploiting the parallelism provided by multiple or parallel servers. Further work is required on how parallel or distributed database systems can be used to improve capacity and performance, such as, extension of the data distribution strategy to provide dynamic data reorganization, and caching the results of database queries.

Another aspect of electronic commerce applications that requires further investigation is how dynamic Web pages may be efficiently managed. These pages are often generated because dynamic data stored in the database are used in their construction. Caching of dynamic Web pages necessitates tracking the dependencies between the data stored in the database and the copies in the cache. There is a need to understand how data dependencies may be tracked without impeding system scalability.

For dynamic data, a change in the database content may result in the corresponding dependent pages becoming stale. Further research is required to understand staleness of data in this context. Moreover, the current trend in providing service to users is toward personalization and choice. This would make more Web pages dependent on users' personal preferences, and pose challenges to data management and performance improvement techniques such as caching.

In our multiple cluster architecture, a data item at the secondary cluster becomes stale (or outdated) when the same data item is updated at the primary. Staleness can be eliminated if the primary cluster notifies the secondary cluster immediately upon completion of each update.

ACKNOWLEDGMENTS

The research reported in this article is part of a CITR major project, Enabling Technology for Electronic Commerce Applications. Researchers from the University of Waterloo, University of Ottawa, University of Western Ontario, Université du Québec à Montréal, Université de Montréal, and Concordia University participated in this major project. The topics of research included performance and scalability, multimedia catalogs, virtual malls, user interface design, and quality of service management. The industrial partners in the major project were IBM, Mitel Corporation, and Bell Canada.

This research was supported in part by a grant from the Canadian Institute for Telecommunications Research under the NCE program of the Government of Canada and in part by the IBM Centre for Advanced Studies and an IBM Faculty Partnership Award. This article represents the views of the authors and not necessarily those of IBM.

REFERENCES

[1] B. C. Neuman, "Scale in Distributed Systems," *Readings in Distributed Computing Systems*, IEEE Comp. Soc. Press, 1994.

[2] J. Wigglesworth, "Bringing Academic Research Directly to Development: IBM Centres for Advanced Studies," *Proc. IEEE Int'l. Eng. Management Conf.*, Cambridge, UK, 2002, pp. 866–70.

[3] M. Abrams *et al.*, "Caching Proxies: Limitations and Potentials," *Proc. 4th Int'l. World Wide Web Conf.*, 1995.

[4] I. Ahmad *et al.*, "Evolutionary Algorithms for Allocating Data in Distributed Database Systems," *Distrib. and Parallel Databases*, vol. 11, no. 1, Jan. 2002, pp. 5–32.

[5] M. Colajanni, P. S. Yu, and D. M. Dias, "Analysis of Task Assignment Policies in Scalable Distributed Web-Server Systems," *IEEE Trans. Parallel and Distrib. Sys.*, vol. 9, no. 6, June 1998, pp. 585–600.

[6] M.-V. Salem, J. W. Wong, and G. V. Bochmann, "A Scalable Load-Sharing Architecture for Distributed Applications," *Proc. IEEE SoftCom 2001*, pp. 156–64.

[7] M. K. Y. Au *et al.*, "Method and Apparatus for a Distributed Web Commerce System," U.S. patent app. 20020174034, Nov. 21, 2002.

[8] J. W. Wong, D. Evans, and M. K. Kwok, "On Staleness and the Delivery of Web Pages," *Proc. CASCON 2001*, Toronto, Canada, Nov. 2001, pp. 156–64.

[9] A. Dingle and T. Partl, "Web Cache Coherence," *Proc. 5th Int'l. World Wide Web Conf.*, 1996.

[10] H. Ye *et al.*, "Towards Database Scalability through Efficient Data Distribution in E-commerce Environments," *Proc. 3rd Int'l. Symp. Elect. Commerce*, Research Triangle Park, NC, 2002.

BIOGRAPHIES

GREGOR V. BOCHMANN [F] (bochmann@site.uottawa.ca) is a professor at the School of Information Technology and Engineering at the University of Ottawa since January 1998, after 25 years at the University of Montreal. He is a fellow of ACM and a member of the Royal Society of Canada. He is known for his work on communication protocols and software engineering. Ongoing projects include distributed network management and quality of service negotiation for distributed multimedia applications.

JOHNNY W. WONG [SM] (jwwong@uwaterloo.ca) received his Ph.D. degree in computer science from the University of California at Los Angeles in 1975. Since that time, he has been with the University of Waterloo where he is currently a professor and director of the School of Computer Science.

TERENCE C. LAU (lautc@ca.ibm.com) is a senior research staff member at the Centre for Advanced Studies (CAS), IBM Canada, and an adjunct professor of the Department of Electrical and Computer Engineering, University of Waterloo. Previously, he was a senior system architect and development manager in various IBM product groups in e-commerce, data communications, imaging, and application development. He received a Ph.D. in computer science from the University of Waterloo.

DON BOURNE (dbourne@ca.ibm.com) received his B.A.Sc. from the University of Toronto in engineering science in 1994, and has worked at the IBM Toronto Laboratory since 1996. Significant areas of contribution include dynamic web page caching and automatic page invalidation, scalable commerce systems, and problem determination/problem source identification. He is currently working as the RAS architect for IBM's WebSphere Application Server group.

DAVID EVANS [StM] (dfevans@bbcr.uwaterloo.ca) received his B.Sc. in computing and information science from the University of Guelph, followed by an M.Math from the School of Computer Science at the University of Waterloo where he is currently a Ph.D. candidate. His research interests include self-organization, performance analysis, and multicast as applied to IP-centric networks and applications.

BRIGITTE KERHERVÉ (kerherve.brigitte@uqam.ca) is an associate professor in the Department of Computer Science at Université du Québec à Montréal, Canada. She received her Ph.D. in computer science from the Université Paris VI, France, in 1986. Her research interests include quality of service management, metadata for multimedia databases, adaptive video delivery, as well as advanced database systems to support distributed multimedia applications.

MOHAMED-VALL M. SALEM (salem@iro.umontreal.ca) is currently a postdoctoral researcher with the School of Information Technology and Engineering at the University of Ottawa. His research interests are in distributed systems, computer networks, and software engineering, and encompass scalability, content distribution, and performance analysis. He received a Ph.D. in computer science from the University of Montreal in 2002. During his Ph.D. studies he held a research fellowship from the IBM Center for Advanced Studies.

HAIWEI YE (ye@iro.umontreal.ca) is currently a Ph.D. candidate in the Department of Computer Science and Operational Research at the University of Montreal. Her research interests are mainly in the areas of quality of service, distributed systems, network management, enabling technology for e-commerce applications, and various aspects of database technology, particularly distributed query processing and optimization.