# The Raincore API for Clusters of Networking Elements

**Chenggong Charles Fan** • *Rainfinity*
**Jehoshua Bruck** • *California Institute of Technology*

The rapid growth of the Internet over the past several years has focused on speed of deployment while, in many cases, ignoring the requirements of high-performance end-to-end transactions. Given that the Internet is primarily an infrastructure to flow information from where it is stored to where it is requested, we can view the communication path between end points as a chain and each networking device along the path as a link in the chain, as shown in Figure 1.

So how can we make this chain stronger?

Clustering technology offers a way to increase overall reliability and performance by strengthening one link in the chain without adding others. We have implemented this technology in a distributed computing architecture for network elements. The architecture, called Raincore, originated in the Reliable Array of Independent Nodes, or RAIN, research collaboration between the California Institute of Technology and the U.S. National Aeronautics and Space Agency's Jet Propulsion Laboratory.[1] The RAIN project focused on developing high-performance, fault-tolerant, portable clustering technology for spaceborne computing (see the sidebar, "RAIN and Other Related Work in Cluster Technology," p. 72). The technology that emerged from this project became the basis for a spinoff company, Rainfinity, which has the exclusive intellectual property rights to the RAIN technology.

In this report, we describe the Raincore conceptual architecture and distributed services, which are designed to make it easy for developers to port their applications to run on top of a cluster of networking elements. We include two applications: a Web server prototype that was part of the original RAIN research project and a commercial firewall cluster product from Rainfinity.

## Distributed Systems in a Networking Environment

A cluster of networking elements operating together for the same purpose effectively creates a distributed system in a networking environment. For example, the front end of the server farm in Figure 1 could use a cluster of firewalls rather than a single firewall.

The objective of a distributed system is to

- balance the processing load by distributing network traffic among the member nodes in a way that increases overall throughput, and
- enable healthy nodes to discover failed nodes and to take over their networking traffic without interrupting the traffic flow.

The key challenges of a distributed system are to maintain consensus among the machines on the exact state of the cluster and to make collective decisions without conflicts.

Networking environments pose three unique requirements on distributed system solutions:

- the need to scale up networking throughput as well as computing power,
- the need to compensate for the negative performance effects of task switching between the different services supported by networking elements, and
- the need for fast fail-over time to maintain network connections in the event of failures.

Raincore is designed to meet these challenges for Internet applications at corporate firewalls and other gateways. It manages load balancing and node failover and helps applications to share state over a cluster of computing nodes. It scales horizontally without introducing additional hardware layers. Furthermore, multiple Internet applications can coexist with Raincore on the same group of physical computers. This reduces the number of links in the Internet chain, thereby improving overall Internet reliability and performance.

## Raincore Architecture and Software Development Kit

Raincore is first and foremost a distributed network computing architecture. It reflects the convergence of computation and communication by building distributed computing protocols into the communication stack.

As shown in Figure 2, Raincore protocols and services are mapped into Layer 4 through Layer 7 of the Open Systems Interconnect (OSI) networking model. The protocols and services are designed to help an application share the traffic load among the cluster nodes and mask failures so that they do not affect the availability of the overall network service. We have published more detail on their design elsewhere.[2]

### Transport Manager

The Raincore Transport Manager is a module that sits at the bottom of the Raincore stack. TM requires the availability of an unreliable unicast interface to send and receive packets. In typical implementations, it uses either a user datagram protocol (UDP) or a raw socket as the packet sending and receiving interface, but it can use any packet-sending interface. The TM can be used to communicate between nodes over a LAN or WAN; there is no limitation on the Layer-2 protocols.

Like transmission-control protocol (TCP), Raincore TM provides a reliable unicast transport to the upper layer. However, Raincore TM provides additional functionalities that are not available in TCP:

- Atomic connectionless message delivery. A packet is either completely delivered or not delivered at all. TM does not employ the concepts of connections or streams. Hence, there is no connection state information to track as nodes go up and down.
- Notification to the upper layer both when it receives acknowledgment from the destination and when all sending efforts have failed. The failure-in-delivery notification serves as a local-view failure detector for the Raincore Group Communication Manager service.

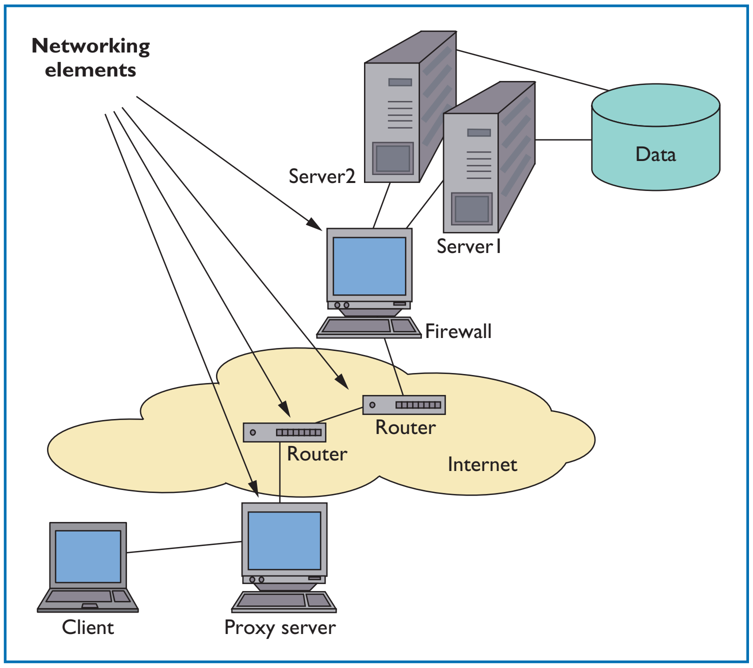


*Figure 1. A traffic path between a client and a server. Clustering technology can be applied to any networking element on this path.*

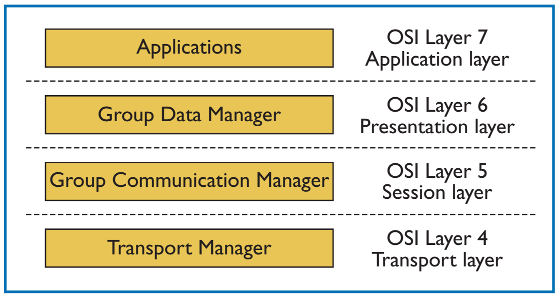


*Figure 2. Raincore network computing architecture. Transport Manager performs reliable unicast communication; Group Communication Manager provides reliable multicast communication; Group Data Manager manages shared data items.*

- Communication between nodes using multiple physical addresses and automatic resend of a packet if earlier delivery attempts failed. TM will exhaust all configured resends on all physical addresses before sending the failure notification to the upper layer. This enables redundant links between the cluster nodes, thereby making the cluster more resilient to link failures and less likely to being partitioned. User applications can specify a packet-sending strategy that targets multiple physical addresses in sequential or parallel order.

When the upper layer calls the Raincore TM to send a message to another node, the TM partitions the message into packets. The packet size depends

## RAIN and Other Related Work in Cluster Technology

Raincore originated in a research project that addressed distributed computing and data storage systems for future spaceborne missions for NASA. The Reliable Array of Independent Nodes project identifies key building blocks for reliable systems built with inexpensive off-the-shelf components. The RAIN testbed includes blocks for reliable communication, group membership information, and reliable storage implemented on a heterogeneous cluster of computing and storage nodes. The system can tolerate multiple node, link, and switch failures, with no single point of failure. In addition, the RAIN architecture supports efficient use of network resources.

The RAIN software components run in conjunction with operating system services and standard network protocols. The technology has been transferred to Rainfinity (http://www.rainfinity.com), which focuses on creating clustered solutions for improving the performance and availability of Internet data centers.

### High-Availability Networking Solutions

Current solutions to the problem of high availability in the networking world include a popular technique called *hot standby*. In this solution, a secondary device is set up to be identical to the primary device. When the primary device fails, the secondary device detects the failure and activates itself to perform the primary device's function. Cisco applies this technique, for example, in its hot standby router protocol (HSRP), which can be used to configure a pair of routers.

Hot standby is a very practical solution to the single-point-of-failure problem, but it does not increase the primary device's performance and therefore does not address the networking performance problem.

*Load balancers* offer another solution. When a device is a single point of failure and a performance bottleneck, you can set up more devices to perform the same function, along with external load balancers at each subnet to which these devices connect. Because the load balancers can both balance the load to and detect the failures among these devices, they address both the reliability and performance problems. Example products include the Nortel Alteon and Cisco Arrowpoint Web switches.

There are limitations to this approach. For one thing, load balancers become additional links in the communication chain, and can become points of failure and performance bottlenecks themselves. Furthermore, load balancers cannot share application state among the devices.

### Distributed Computing Solutions

Distributed computing technology offers some solutions to these problems. Powerful multicomputer systems have achieved performance not possible with a single computer. Prominent examples include the Network of Workstations (NOW) project at Berkeley,[1] the Beowulf project at NASA,[2] and the Shrimp project at Princeton.[3] These systems can bundle tens, even hundreds, of cheaper computers to create a powerful computing system.

Meanwhile, server clusters have emerged to take advantage of the results in distributed computing. These systems are motivated by the need to scale up server performance in the client-server world, where one server usually needs to serve a large number of clients simultaneously. Server clusters address this asymmetry by using multiple computers to create a more powerful logical server. SunCluster[4] from Sun Microsystems and Microsoft Cluster Server[5] are examples of such systems.

Raincore differs from these technologies in its specific focus on networking applications, which require networking throughput to scale up along with computing power.

#### References

1. T.E. Anderson et al., "A Case for Networks of Workstations: NOW," *IEEE Micro*, vol. 15, no. 1, Feb. 1995, pp. 54-64.
2. D.J. Becker et al., "Beowulf: A Parallel Workstation for Scientific Computation," *Proc. 1995 Int'l Conf. Parallel Processing*, 1995; available online at http://www.beowulf.org/papers/.
3. S.N. Damianakis et al., "Client-Server Computing on Shrimp," *IEEE Micro*, vol. 17, no. 1, Jan/Feb 1997, pp. 8-18.
4. Sun Microsystems, "The Sun Enterprise Cluster Architecture," white paper, Oct. 1997; available online at http://www.sun.com/software/whitepapers.html#cluster.
5. Microsoft, "Windows NT Clustering Architecture," white paper, 2001 (updated); available online at http://www.microsoft.com/ntserver/productinfo/enterprise/clustering/ClustArchit.asp.

on the packet-sending interface and the Layer-2 protocol that the TM is using. For example, to send UDP packets over Ethernet, keeping the packet size under 1.5 Kbytes makes UDP packet fragmentation less likely. TM employs a sliding-window protocol to provide reliable delivery and flow control. The receiving side won't pass the message to the upper layer until all packets related to that message have been received.

TM allows multiple upper layer users and distinguishes them by assigning different channel IDs to different users. Multiple TMs can also coexist on the same physical node by using different ports.

There are five key TM functional interfaces:

- TM_addTarget() — registers a target with the TM.
- TM_getLogicalTime() — returns the logical time. The time values can be used for partial ordering. For example, if event e2 is causally related to event e1, logicalTime(e2) > logicalTime(e1). The logical clock does not help with concurrent events. Where it matters, the Group Data Manager uses the TM's logical clock and provides a consistent total ordering of events.
- TM_registerChannelInfo() — registers channel-specific information. The TM can service multiple clients and each client uses a cluster-wide unique channel to communicate with its counterparts on other cluster targets. This function lets the client handle how to configure messaging.
- TM_unicastReliableMessage() — sends a reliable message to the given target.
- TM_unicastUnreliableMessage() — sends an unreliable message to the given target.

### Group Communication Manager

Group communication for a distributed system in any environment must provide a reliable and efficient many-to-many communication transport that will continue to function as nodes leave and join the group, or cluster. We characterize this functionality to be session-layer functionality and created the Group Communication Manager module to carry it out.

The GCM maintains consistent group membership and provides reliable multicast transport for sharing application state among the cluster group and for facilitating transparent failover of traffic from a failed node to a healthy node. GCM sits on top of the TM. Multiple GCMs can coexist on the same node and use different channels of the same TM.

Key functions in the GCM include:

- GCM_queryCommunicationMembership() — obtains from GCM a description of the current communication group, including all the nodes that are currently up and communicating among the cluster.
- GCM_queryEligibleMembership() — obtains from GCM a description of the current eligible group, including all nodes that are eligible to be part of the cluster.
- GCM_setParameters() — changes the values of GCM parameters, including the pointers to various notification callback functions.
- GCM_getParameters() — obtains the current values of GCM parameters.
- GCM_acquireMasterLock() — acquires the GCM master lock.
- GCM_releaseMasterLock() — releases the GCM master lock.
- GCM_sendReliableMessage() — requests GCM to reliably send a message to one or more target nodes.
- GCM_sendUnreliableMessage() — requests GCM to unreliably send a message to one or more target nodes.

### Group Data Manager

The Raincore Group Data Manager is implemented on top of the GCM as a presentation-layer service to the applications. It manages *data items* for the applications; the application can *read* from and *write* to these data items, which are mirrored on all nodes in the cluster. The GDM automatically synchronizes any modification to the data items by any nodes. It uses the GCM to multicast the changes to other nodes. The underlying communication complexity is hidden from GDM users.

The GDM implementation includes a global clock, implemented under Lamport's guidelines. It is not a physical clock, but it provides consistent ordering of events in a cluster. (Although the ordering may be incorrect, at least all nodes agree to it.) The clock is used to resolve conflicts that result when two nodes try to write to the same data item. The GDM can maintain many data items of various sizes. It provides interfaces that give users the option of treating the data item as an atomic unit or modifying parts of it.

The GDM includes a *distributed lock manager*. A user can create locks to associate with one or more data items. The locks are, in fact, data items themselves. The GDM uses the GCM's mutual exclusion service to manage the locks, waiting for the arrival of a token to acquire and release a lock. This can be used to maintain the read-write consistency to the data items.

The GDM also lets users adopt the concept of a *transaction* to access multiple data items atomically. A user may start a transaction and then write to a number of data items. The GDM will not synchronize these changes until the transaction concludes.

The key GDM functional interfaces are as follows:

- GDM_SharedData_new() — creates a new shared data item.
- GDM_SharedData_delete() — deletes the shared data item.
- GDM_SharedData_read() — reads the shared data item.
- GDM_SharedData_write() — writes the shared data item.

- GDM_genUniqueID() — generates a cluster-wide unique identifier (a utility function).
- GDM_registerDeletedDataCallback() — calls the given function whenever a shared data item is deleted.
- GDM_registerNewDataCallback() — calls the given function whenever a shared data item is created.
- GDM_Lock_acquire() — acquires the specified lock and calls back the application when the acquisition is made.
- GDM_Lock_create() — creates a long-duration lock that can be held for an arbitrary amount of time.
- GDM_Lock_destroy() — destroys and frees the specified lock.
- GDM_Lock_release()— releases the specified lock.
- GDM_Lock_wait() — blocks the calling thread on the specified lock (function available only on multithreaded environments).

## Raincore Applications

Raincore has been implemented in both the academic community and industry. The first application, called SNOW (for "Strong Network of Web servers"), is a scalable Web server cluster that was developed as part of the RAIN project. The second application, RainWall, is a commercial solution that provides the first fault-tolerant and scalable firewall cluster.

These applications exhibit the fast failover response, low overhead, and near-linear scalability of the Raincore protocols (see the sidebar, "Raincore Performance").

### SNOW Web Server Prototype

The Raincore protocols were conceived while we were building the SNOW proof-of-concept prototype for the RAIN project. We constructed the prototype in Caltech's Parallel and Distributed Systems (Paradise, for short) Lab. The six nodes consisted of dual Pentium Pro 200-MHz servers, redundantly interconnected by four Fast Ethernet switches. The design goal was to develop a highly available fault-tolerant Web server cluster that would distribute load among the nodes and continue to function when parts of the system failed.

SNOW uses the Raincore Transport Manager to handle all messages passing between the servers. It takes advantage of the redundant interconnections, so that cluster partitioning becomes less likely. It uses the Raincore Group Communication Manager to manage the group membership. The GCM detects node crashes and implements failover.

A multicast model is assumed in the SNOW implementation. The HTTP requests are received by all servers in the cluster. The servers use Raincore's mutual exclusion service to decide which server node will own and reply to that request. Servers other than the designated one will simply drop the request. The assignment of who will own the connection is a function of the load distribution on all the server nodes. The server health and load information, as well as the HTTP request assignment table, are shared among the cluster nodes through an early version of the Group Data Manager.

### RainWall Fault-Tolerant and Scalable Firewall Application

Firewalls offer a single security administration point to control access to and block intruders from enterprise and personal data. Unfortunately, they also become bottlenecks for the organization, customers, and partners trying to access a site. RainWall applies Raincore technology to a high-availability and load-balancing clustering solution for firewalls.

RainWall is typically installed on a cluster of gateways, along with the firewall software, and performs load balancing among the gateways. The software installs on the same machines as the firewall itself and so doesn't introduce any new layers of components in the network. In the event of failure of one or more gateways, RainWall routes traffic through the remaining gateways without interrupting existing connections.

**Virtual IP Manager.** RainWall uses a Virtual IP Manager to manage the pools of virtual IP addresses for the firewall cluster. The virtual IPs are the only advertised IP addresses of the firewall cluster; they are specified in the routers and local clients as default gateways. All traffic that goes through the firewall is being directed to one of the virtual IPs. By managing the virtual IPs intelligently and efficiently, RainWall guarantees firewall availability in the presence of failures and achieves optimal performance even under heavy loads.

In addition to the Virtual IP Manager that provides coarse load-balancing and traffic failover among the firewalls, RainWall includes a kernel-level software packet engine that load-balances traffic connection-by-connection to all firewall nodes in the cluster. This module also provides a way to synchronize connection state information without race conditions. The load and connection assignment information are shared among the cluster nodes using Raincore.

## Raincore Performance

The most important measure of Raincore performance is the scaling in an application's network throughput as the cluster size increases. We have performed this measurement on RainWall.

Figure A presents a benchmark from the Rainfinity Lab. In this test, RainWall ran on a cluster of one, two, and four gateways consisting of Sun Ultra-5 single-CPU 360-MHz workstations in a Fast Ethernet switched environment. Each workstation had 256 Mbytes of RAM and ran Solaris 2.6. HTTP clients are placed at one side to request data from Apache Web servers on the other side of the RainWall cluster. The throughput number on the graph indicates the total traffic traveling through the RainWall cluster. As the figure shows, the throughput scaling from one node to two nodes is 1.97, and the scaling from one node to four nodes is 3.76.

Figure B illustrates the results of a benchmark designed to reveal Raincore's CPU usage exactly. The benchmarking program varies the size of the state being synchronized and measures the CPU usage. In particular, this program uses the Group Data Manager API. The program maintains 10,000 data items among the cluster nodes. We vary the size of the data items as well as the number of data items updated each second. We then measure the CPU overhead on each node in the cluster and calculate the average.

Typically in a networking application, the state being shared is the state associated with each networking session, and the size of data for each session ranges from 100 bytes to 1 Kbyte. Figure B shows that at 1 Kbyte per data item and 500 data items updated per second per node, the CPU overhead is still below 1 percent.

Figure B also shows that for a networking application using Raincore, if 1 Kbyte of data is used to record the information about each session, a two-node Raincore cluster can accept 1,000 new sessions per second and still not encounter significant CPU overhead for the state synchronization between cluster nodes. This translates to 3.6 million new sessions per hour, or 86.4 million new sessions per day.

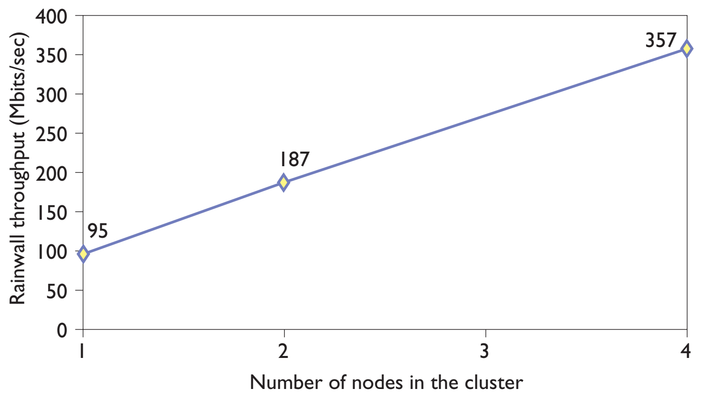
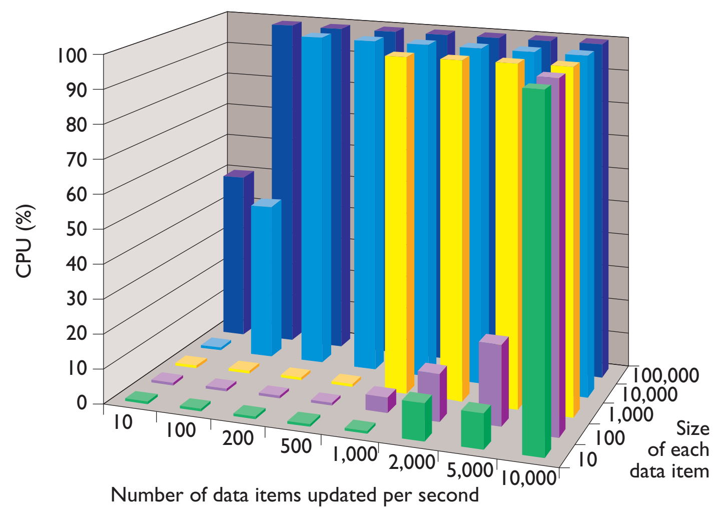


*Figure A. RainWall performance benchmark. HTTP clients were placed at one side to request data from Apache Web servers on the other side of a RainWall cluster of one, two, and four nodes. The results show almost linear scaling; CPU usage was less than 1 percent.*



| | 10 | 100 | 200 | 500 | 1,000 | 2,000 | 5,000 | 10,000 |
|---|---|---|---|---|---|---|---|---|
| 🟩 10 | 1 | 1 | 1 | 1 | 1 | 12 | 10.5 | 100 |
| 🟪 100 | 1 | 1 | 1 | 1 | 1 | 14 | 25.5 | 100 |
| 🟨 1,000 | 1 | 1 | 1 | 1 | 100 | 100 | 100 | 100 |
| 🟦 10,000 | 1.5 | 47.5 | 100 | 100 | 100 | 100 | 100 | 100 |
| 🟦 100,000 | 51.5 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |

*Figure B. Two-node Raincore Windows NT benchmark. Each node is a dual-CPU Pentium III 933-MHz computer, with 128 Mbytes of RAM. The number of data items updated per second per node varies from 10 to 10,000, and the size of the data item varies from 10 bytes to 100 Kbytes. The results show the percentage of CPU overhead given different state synchronization rates in Raincore. The results validate that Raincore can handle substantial application traffic (for example, 3.6 million new sessions per hour) without encountering a significant overhead.*

**Local Failure and Fault Detection.** RainWall includes local failure detectors that constantly examine whether required local resources are functioning correctly. The detectors examine three required components:

- the network interface cards for link connectivity,
- the firewall software for proper function, and
- the local machine's ability to reach remote hosts via ping.

If any of these resources go down, RainWall will, by default, bring down that firewall node, and reassign its virtual IP addresses to other healthy firewall nodes with no interruption of service.

The local failure detectors, working together with Raincore, go a long way in handling faults in today's networking environment. A local fault monitor also serves to prevent partitioned clusters, usually a challenging problem for distributed systems. The failover time of RainWall is on average 2 to 3 seconds. For example, if a client is downloading a file from a server through a firewall and if a network cable connecting one of the RainWall firewalls is accidentally unplugged, the client, instead of losing the connection, will see only a 2- to 3-second hiccup in the traffic flow before it fully resumes. This is well within the TCP time out, so all TCP connections can failover transparently.

**Current Implementation.** The core RainWall code is written in C/C++ and is compiled to native code for efficiency. It has been implemented for Solaris, Windows NT, and Linux. RainWall runs as a user-space process and a kernel module. The Raincore portion was implemented in the user-process level.

From RainWall's graphical user interface, a user can monitor the health of each firewall node, the load going through each node, and the virtual IP address assignment in the cluster. The administrator can set virtual IPs to be sticky, so they stay on particular nodes and don't participate in load balancing. Virtual IPs can be preconfigured with a preference for particular machines, and users can drag-and-drop them between machines through the GUI. All state information is shared using Raincore protocols.

RainWall is operational at more than 200 major customer sites worldwide.

## Conclusion

As the Internet continues to grow, the Raincore API gives developers a powerful software toolkit to help solve high-availability and load-balancing problems. It applies to a wide array of networking applications, such as firewalls, virtual private networks (VPNs), IP telephony gateways, application routers, and Web servers. It is not, however, designed for back-end shared-storage database server applications, where server-clustering software such as SunCluster and Microsoft Cluster Server offer a better fit.

Rainfinity is currently focused on extending the Raincore technology to more advanced topologies, implementing it at lower hardware layers, and extending the API to address emerging applications in the storage and wireless space. We are also examining its implementation on top of emerging cluster technology standards, such as the Virtual Interface Architecture (http://www.viarch.org) and Infiniband (http://www.infinibandta.org).

For more information, see the company's Web site at http://www.rainfinity.com.

## References

1. V. Bohossian et al., "Computing in the RAIN: Reliable Array of Independent Nodes," *IEEE Trans. Parallel and Distributed Systems*, vol. 12, no. 2, Feb. 2001, pp. 99-114.
2. C.C. Fan and J. Bruck, "The Raincore Distributed Session Service for Networking Elements," *Proc. Int'l Parallel and Distributed Processing Symp., Workshop on Communication Architecture for Clusters*, IPDPS, http://www.ipdps.org, 2001; available online at http://vigeland.paradise.caltech.edu/~fan/research.html.

**Chenggong Charles Fan** is a cofounder and director of engineering at Rainfinity. His research interests are in distributed fault-tolerant computation and communication. He received his BE from Cooper Union in 1995, summa cum laude in electrical engineering, and his MS and PhD from the California Institute of Technology in 1996 and 2001, respectively, also in electrical engineering.

**Jehoshua Bruck** is the Gordon and Betty Moore professor of computation and neural systems and electrical engineering at the California Institute of Technology. He is also a cofounder and chair of Rainfinity. He received a BSc and MSc in electrical engineering from the Technion, Israel Institute of Technology, in 1982 and 1985, respectively, and a PhD in electrical engineering from Stanford University in 1989. Bruck is a recipient of a 1997 IBM partnership Award, a 1995 Sloan Research Fellowship, and a 1994 National Science Foundation Young Investigator Award. He holds 22 patents and is a Fellow of the IEEE.

Readers may contact the authors via e-mail at fan@rainfinity.com and bruck@paradise.caltech.edu.