SRIRAM: A scalable resilient autonomic mesh

by D. C. Verma
S. Sahu
S. Calo
A. Shaikh
I. Chang
A. Acharya

Mirroring and replication are common techniques for ensuring fault-tolerance and resiliency of client/server applications. Because such mirroring and replication procedures are not usually automated, they tend to be cumbersome. In this paper, we present an architecture in which the identification of sites for replicated servers, and the generation of replicas, are both automated. The design is based on a self-configuring mesh of computers and a communication mechanism between nodes that operates on a rooted spanning tree. A query-search component uses Java™ language-based query capsules traveling along the branches of the spanning tree, and a caching scheme whereby the query and previous search results are cached at each node for improved efficiency. Furthermore, a security and anonymity component relies on one or more authentication servers and an anonymous communication scheme using link local addresses and indirect communication between the nodes via the spanning tree. The architecture also includes components for resource advertising and for application replication.

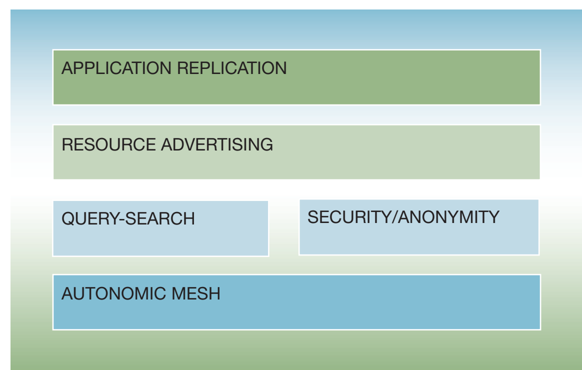
Most networked applications are currently implemented using a client/server computing model. A server with a well-known address hosts the application, while different clients access it over the network. Typically, the server (or a set of servers) will be located at a single site, and the overall performance of the application will depend upon factors such as the speed of the network between the client and the server, the computing power at the hosting site, and congestion in the network. The concentration of servers at a single site also reduces the ability of the application to withstand failures. In order to improve the availability and reliability of a system, distributed architectures incorporating replicas and mirrors are frequently used. However, the process of replication and mirroring is usually manual and, due to the complexity in the control and management of the system, somewhat cumbersome. An autonomic replication and mirroring facility would significantly simplify the process of replication and would improve the availability of applications.

In this paper, we describe the highly scalable distributed architecture SRIRAM (Scalable Replication Infrastructure using Resilient Autonomic Meshes), which is designed to dynamically create replicas of applications for resilient operation. The basic idea behind SRIRAM is that several computers are available at any given time on the network, and an application deployed on one of the machines can be mirrored and run on any other machine that is available and capable of providing the same service. All the computers are connected in a mesh with self-managing properties. A machine hosting an application uses the communications overlay (an application-level communication network that overlays the mesh) to transmit the application's replication

©Copyright 2003 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

0018-8670/03/$5.00 © 2003 IBM

Figure 1 The SRIRAM architecture



requirements, identify potential replicas, and configure the replicas to start a copy of the application. Clients search for one of the replicas of any application in which they are interested, and invoke the services of that application from the replicated copy.

The autonomic replication infrastructure provided by SRIRAM can be used in various scenarios, but is most relevant in the context of peer-to-peer networks,[1,2] content distribution networks,[3,4] and Grid computing.[5] The basic SRIRAM architectural components can be used to improve the underlying communication infrastructure (resiliency to faults, increased availability, etc.) in each of these contexts, with the application replication mechanism as a specific feature provided within each of these operating environments.

The rest of the paper is structured as follows. In the next section we give an overview of the SRIRAM architecture. In the following four sections we discuss each of the major components of the architecture as well as the types of applications that can exploit the replication support provided by SRIRAM. In the remaining two sections we review related work, and then we present our conclusions and directions for future research.

Architecture overview

The SRIRAM architecture, which includes five major components, is illustrated in Figure 1. The mesh creates a network interconnecting all machines participating in the system. A flexible and efficient query-search mechanism is built on top of the network. Security and anonymity controls round up the communication infrastructure consisting of the bottom three components. The upper two layers are a specific use of this infrastructure. The query-search mechanism facilitates the resource advertising by the participants on the mesh. The resource advertising facility is used for automatic search and for creation of replicas.

The *autonomic mesh* component consists of a self-configuring network interconnecting all machines in the system. This layer supports a broadcast mechanism that allows all the participating machines to communicate in an efficient, scalable, self-configuring, and self-healing manner.

The *query-search* component supports basic search primitives that allow participants to search for information about other participants within the SRIRAM system. SRIRAM uses a system based on active programs (Java** language-based query capsules), which enables a flexible and efficient search mechanism. Caching is used to improve the responsiveness of the system.

The *security/anonymity* component provides for communication with other peers while preserving the anonymity of the requester or the respondent. Security and access control within SRIRAM is based on digital certificates issued by trusted authentication servers. The *resource advertising* component allows a participating machine to describe the resources required for replicating the applications running on it, and for possible replicas to indicate their resource availability.

Finally, the *application replication* component provides the basic functions for replicating the code and data of applications, and for maintaining the proper consistency of application data among the different mirrors. Each one of these components is described in more detail in subsequent sections.

Autonomic mesh algorithm

The autonomic mesh component within SRIRAM provides an overlay that interconnects all of the participating machines so that they may communicate with each other. This function is similar to the overlays created in distributed peer-to-peer networks like Gnutella[2] that enable group communication among all participants. However, Gnutella and similar systems use a flooding scheme for their group communication, which consumes a significant amount of network and node resources. In SRIRAM, we have opted

for a scheme that builds a rooted spanning tree among all the participants and attempts to minimize the number of messages exchanged for any given query.

The use of a rooted spanning tree has its own set of problems. The traditional distributed algorithms for creating a spanning tree are relatively slow and complex, and are thus impractical for our needs. Furthermore, nodes that are nearer the root of the rooted spanning tree are likely to see more traffic than nodes at the leaves of the tree. The tree is also more likely to be disrupted when a machine leaves or joins the system.

To accelerate the process of spanning tree creation, SRIRAM uses a semi-distributed scheme similar to that used in YOID.[6] In the semi-distributed scheme, SRIRAM deploys a number of hint-servers within the system. The hint-servers store a limited amount of information about the participants, and the information is not guaranteed to be up-to-date. A participant wishing to join the system communicates with the hint-servers in order to obtain the identities of possible nodes in the existing spanning tree to which it can connect.

To solve the problem of increased load on participants near the root of the tree, a ranking scheme is used. Each participating node in SRIRAM computes a rank for itself. A rank is a measure of the computing capability of the node. For computational ease, the closer the node is to the center of spanning tree activity, the lower its rank. The root of the spanning tree is the node with the lowest rank. In addition, the rank computation also involves the inverse of a weighted combination of its CPU speed, available disk space, memory size, and speed of its network interfaces. Ranks impose a strict ordering on the participants in the tree, and ensure that no cycles can form in the constructed tree.

For efficient tree creation and restructuring, a new machine is only allowed to join the tree by choosing a parent from among existing participants with ranks lower than itself. This provides a simple, yet effective, scheme for eliminating cycles in the spanning tree. The selected participant becomes the parent of the new node. When a participant leaves the tree, its children join the parent of the departing machine. If an orphaned child node does not succeed in joining any node, it then increases its own rank and contacts the hint-server for a list of possible parents. The steps in the creation of the spanning tree are described below in further detail.

Joining the tree. When a new machine is about to join a tree (this is known as the registration phase), it computes its rank and contacts the hint-server to obtain a list of machines with ranks slightly lower than the computed rank. It then contacts each of the machines in the list and requests that it become its parent. A machine in the list may accept the request only if it has a lower rank than the newcomer. In addition, it may refuse to accept new children beyond a certain preconfigured limit, or it may no longer be up. The delay in obtaining a response from the machine is used to estimate the round-trip delay between the potential parent and the newcomer. The newcomer joins (as child) the machine with the lowest latency that responds positively to its join request. If no machine in the list responds positively, the newcomer doubles its computed rank and obtains a new list from the hint-server. If a newcomer has a rank smaller than the current root, the hint-server returns a special code to both the current root machine and the newcomer, asking that the newcomer become the new root of the spanning tree as the parent of the existing root machine.

Tree improvement algorithm. The node that a newcomer initially selects as its parent may not be the best choice for the system. In order to continually improve the structure of the spanning tree, each node periodically obtains a list of its siblings (the other children of its parent) and the name of its grandparent (the parent node of its parent node). It then assesses the latency and the feasibility of these machines to become its parent. If a machine with lower rank and latency is found, then the node switches over to the new parent. The tree improvement process is an ongoing procedure that tries to optimize the spanning tree configuration, which—given the dynamic arrival and departure of nodes—could otherwise deteriorate over time.

Data structures at the hint-servers. Each hint-server in SRIRAM maintains a data structure containing a partial list of the current participants in the system. The participants are maintained in a fixed-size list, sorted according to their ranks. A participant is first entered into the list when, as a newcomer, it queries the hint-server for joining the tree. During the registration phase, each participant indicates the number of children it is able to support. If the number of children is nonzero, the participant is entered into the list. If the capacity of the list is exceeded, the

participant with the highest rank is removed from the list. At the time of a join request, a set of K participants is randomly selected from the next 2K participants with rank higher than the newcomer and returned to the newcomer. Here K is a configuration parameter for the hint-server, with a default value set to be the smaller of 10 and one-hundredth of the number of participants in the system. After a participant's name has been given out more than 2K times, it is removed from the data structure.

The hint-server does not keep track of the nodes' status as they join or leave the system. Therefore, the information provided by the hint-server may be out-of-date, and the participants use the schemes described previously to work around such inaccurate information. Since there is no need to maintain consistent information, a single hint-server can easily support thousands of participants with the only constraint being the amount of storage set aside for its data structures.

Handling tree partitioning. Partitioning of the tree is handled by a relatively simple scheme. Each node maintains the identity of the root of the tree to which it belongs. Each root node periodically sends out a message broadcasting its identity along the branches of the tree. The identity of the root is compared in messages exchanged to monitor response times in the tree improvement algorithm. When a node detects that another node in the system has a different notion of the identity of the root node, a root conflict resolution message is sent up to the parents of each node. The root conflict resolution continues up to the roots of the two trees, and the two roots join together with the higher ranked root becoming a child of the lower ranked root. The ranking criteria used by SRIRAM have the effect of establishing well-connected, more powerful nodes as the root node. The root conflict resolution messages are expected to be generated relatively infrequently.

Self-configuration of the autonomic mesh. For proper operation of the autonomic mesh, each participant node needs to obtain values for a number of configuration parameters. Examples of such parameters are the frequency at which each node probes its neighbors for tree climbing, the weights used to compute the rank of a participant, and the maximum lifetime of a message sent on the mesh. Other components of the system described later in the paper also need specific configuration parameters, for example, the types of query-capsules that are defined within the system, the identity of certificate servers, and so on. Although each node could choose its own value for a configuration parameter, this would make the entire process more complicated and more difficult to manage.

In order to automate the configuring process, it is assumed that the configuration parameters of an operational SRIRAM system are managed at a central point by an "operator" of the system. The operator maintains a copy of the configuration at the hint-server, and signs it using its public key. The public key and the identity of the operator are available in the digital certificate of the operator. Each participant can obtain a copy of the configuration from the hint-server. The owner of each participant is free to modify the values of its parameters. Alternatively, when a participant attaches, as child, to a new neighbor on the spanning tree, it can obtain the configuration information from the neighbor, validate the signature of the operator, and then use the configuration. The configuration distribution process makes the participating node largely self-configuring (except for those participants who wish to override the operator-specified configuration).

Query-search mechanism

A client node that wishes to locate a resource, whether a file or a service, sends a query along the spanning tree. The query is encapsulated in a query capsule, which is a piece of Java code that, when executed, will match the search criteria provided by the client node against the resources located at the node on which it is being executed. This query capsule is propagated by each node along all the branches of the spanning tree (except the one that sent the query) and executed at each node it traverses. When a node finds a match for the query, it sends a positive response containing its location back along the spanning tree toward the client node. Each intermediate node that receives this positive response caches both the query and the location of the resource so subsequent searches for the same resource will receive a speedier response. If an intermediate node receives multiple positive responses, it may choose to cache some number of them and return the list in response to subsequent queries.

This concept of query capsules is borrowed from active network schemes and permits the formulation of generic queries. Unfortunately, allowing query capsules to execute on nodes presents both security and performance issues. SRIRAM handles this by providing a set of standard query capsules and by al-

lowing each node to restrict the execution of other arbitrary query capsules. The standard query capsules can contain both simple query capsules and complex query capsules. A simple query capsule may search the contents of a file looking for a key word match, whereas a complex query capsule may use more sophisticated techniques to search for resources that provide a specific Web service, for example. It is assumed that all nodes will permit execution of the standard query capsules, so a client node wishing to conduct a search using one of these standard query capsules need only provide the parameters to that capsule in its search query.

Once a positive response has been cached at an intermediate node, any subsequent queries arriving at the node will first be checked against the cache. If the query capsule appears in the cache, or the query can be answered using the results of a query found in the cache, then this node will contact the node listed in the cache as the location of the resource, in order to ensure the information is still valid. If the cache entry is still valid, the intermediate node will send a positive response containing the location of the resource to the client that originated the search and stop further queries from flooding the spanning tree. For popular search topics, caching can thus considerably reduce the number of search messages and save both network usage and query capsule executions at various nodes.

When a client receives a positive response to its query, it may contact the node offering the resource either directly or indirectly to obtain the needed files or invoke the desired service. When the resource provider node is contacted directly, the client may be asked to authenticate itself by providing a certificate issued by the authentication server. Anonymity on queries is also supported, as described in the section "Security and anonymity."

For a simple example of a query search, see Figure 2. N_1 through N_7 are nodes in the spanning tree. The client node (N_1) sends a query capsule up the tree (see path Q) and this query capsule is executed at each intermediate node until node N_5 finds a match. Node N_5 generates a positive response containing its location and sends it back along the tree toward the client node N_1 (see path R). At each intermediate node (N_3 and N_2) the query capsule and resource location are cached before being forwarded. Once the client node receives this positive response to its search, it is free to directly (or indirectly) contact node N_5 and request the desired resources.

Figure 2 Query-search example

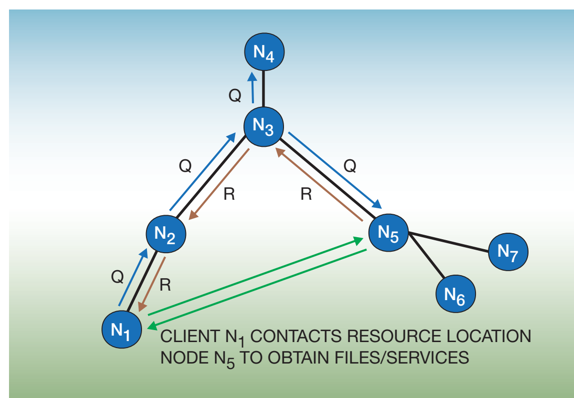
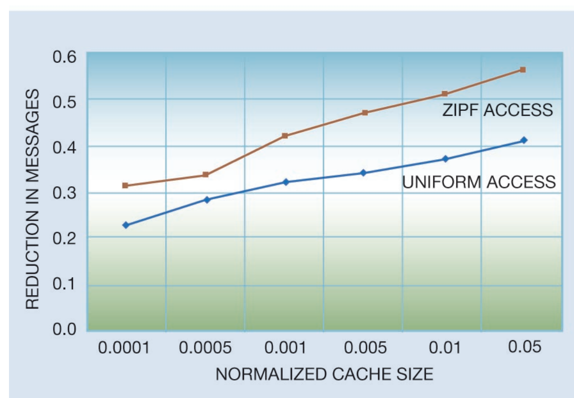


Figure 3 Benefit of query caching



The dynamic improvement of the spanning tree can result in the creation of temporary cycles in the system. In order to eliminate endless looping of queries, each query capsule starts with a preset time-to-live counter, which is decremented every time the query is forwarded by a participant, with the query being discarded once the time-to-live counter reaches zero.

In Figure 3 we present preliminary simulation results illustrating the benefits of caching query results at intermediate nodes. There are two key questions that must be answered in caching the queries: which nodes should cache the queries, and what replacement policy should control the cache mechanism. A simple LRU (least recently used) scheme has been assumed as the cache replacement policy. We have

considered the following three choices for addressing the first issue: (1) all the intermediate nodes that receive the query result cache the result, (2) only the end point nodes cache the query result, and (3) an intermediate node that receives the query result caches it with a probability p, $0 \leq p \leq 1$. We consider a system with 500 nodes for this illustration and each node is randomly assigned a rank between 0 and 500. We assume that there are one million documents available in the system. We vary the cache size so that its capacity varies from 100 to 50000 query results.

Figure 3 illustrates the benefit of caching queries as the normalized cache size varies from 0.0001 to 0.05. There are two graphs, representing two different document popularity distributions. The normalized cache size is obtained by dividing the cache capacity by the total number of unique documents in the system. On the *y* axis, we have plotted the reduction in the average number of messages required for satisfying a query normalized by the number of messages required without caching. We observe that there is an appreciable reduction in the average number of messages required to locate an object with query result caching, for both Uniform and Zipf (with Zipf parameter = 0.9) document access popularity. We observe that the benefit is higher when the access popularity follows a Zipf distribution. In this evaluation, we have assumed that the cached results are always consistent. However, in a dynamic scenario, in which nodes could leave or join at any time, cached results may not always be consistent. We are currently evaluating how to address this problem.

Security and anonymity

In any distributed system, issues related to security and privacy arise. When participants in a mesh are looking for resources, or advertising the availability of resources and services, it may be desirable to maintain their anonymity, or provide that information to a selected set of participants. SRIRAM uses a certificate-based system[7] in order to support privacy and anonymity in its communications.

SRIRAM supports one or more authentication servers. The authentication servers validate the credentials of a participant and issue to the participant a digital certificate. The certificate, signed with the public key of the authentication server, includes the identity of the participant. A separate certificate identifies the groups to which a participant belongs. The certificates are used as part of the challenge-response system to authenticate participants, following the same schemes used by TLS[8] or IPSEC[9] authentication.

When anonymity is desired, the participants need to hide their IP (Internet Protocol) addresses from other participants. SRIRAM uses a scheme based on link local addresses borrowed from the concept of automatic network routing (ANR) proposed in some broadband communication systems.[10] Each participant assigns a random address to its children and parent. The mapping of the address to the real neighbor in the link is only known to the local node.

Anonymous communication is always indirect, using the spanning tree, rather than direct between the involved parties. Two chains of link local addresses are included in anonymous communication, each chain encoding an anonymous path from one sender to the other. The only exceptions are anonymous queries on the spanning tree, which are used to discover the initial chain of link local addresses to use.

A participant anonymously looking for available resources will send out a query on the spanning tree. Each participant will include the link local address of the neighbor from which it has received the query before forwarding it on to the other branches of the spanning tree. The link local addresses are appended to a growing chain of the path to the requester and form the anonymous path back to the querying participant. When a participant sends a response to a query, it removes the last link local address from the local chain, and sends the message to the neighbor with the specified link local address. These responses are also subject to the reverse path accumulation, and create a reverse path to the respondent.

The anonymous communication process is best illustrated by an example. Consider the system of six nodes shown in Figure 4. Each node assigns link local addresses to members on the spanning tree as indicated by labels in the figure; for example, node B has assigned label 4 to its neighbor A, label 7 to its neighbor C, and label 9 to its neighbor E. Let us consider the case of node A sending out an anonymous query on the spanning tree. It sends the query to node B, which creates a link-local chain beginning with 4 (label assigned to A), and forwards it to its other two neighbors C and E. C appends the local link label of B to the chain, which now becomes 45, and forwards the query to D and F. D appends the local link label of C, and has the accumulated path to the sender of 453.

Assuming that D responds to the sender, it uses the last index 3 to send its response back to B, it strips the index from the end of the path before sending it to C. C notes that the response has come from link local D, creates a reverse path of 1 and uses the remaining path of 45 to propagate the response. The last link local address of 5 indicates that the response should go to B. B uses the remaining path of 4 to send message back to A, and appends 7 to the path to the respondent (which is now 17). A is the final recipient, and knows the path to the respondent (171) without knowing that it is D who responded. For further communication, A can use the path 171 to communicate with D, and D can use the path 453 to communicate with A, each being unaware of the other's identity.

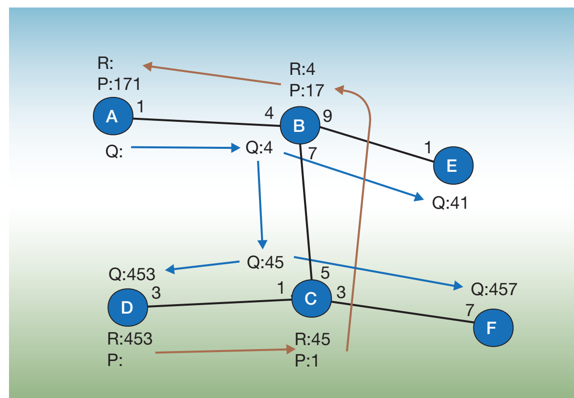
In order for the anonymous communication to work, we only need at least one of the intermediary nodes to play by the rules and not reveal its mapping of link local addresses to others. Each node is aware only of the identity of its immediate neighbors, and is not able to infer the identity of any other participant unless all of the members of the spanning tree along its path collude with it. As the number of participants increases, the ability of any individual to obtain such colluding members becomes negligible.

Resource advertising and application replication

Before a SRIRAM node runs an application (e.g., an instance of a transcoding service), a standardized description of the application to be run should be provided. The description includes the application type (Web server, directory server, Web service and its description) as well as information required for the implementation of the service. That is, a listing of the code and data that are required to host that application is also included as part of the service description, as well as the configuration required for running that application. The service description allows the copying of the desired software and data components, and the launching of another instance of the application to be automated. The service description also includes the scripts, running at the requesting node, that stop and start the application at the machine providing the service.

In addition to the software configuration, the description also includes values for the minimum amount of disk space, the CPU processing power, the network bandwidth, and any constraints on the operating system needed to run that application. The resource advertising module sends out a query capsule on the spanning tree searching for nodes that may be willing to host a replica of the application. The query capsule may be sent anonymously or with credentials, as specified by the configuration file.

Figure 4 Anonymous query search



When a participant receives the query capsule, its resource advertisement module examines the locally available system resources. If the available system resources are suitable for running a replica, and if the machine administrator allows running of replicas of other applications, then the resource advertising module sends a response back to the requester indicating the resources available. The response is sent over the spanning tree for anonymous queries and directly to the requester for nonanonymous queries. The IP address of the respondent is included in this type of response.

The requesting node selects a fixed number of replicas from all the responding participants. The heuristic used gives preference to machines with the largest weighted combination of available resources and the largest absolute numeric difference in IP addresses. The application replication module is then invoked to create a replica of the application.

The replicas are created by copying the contents of the software and configuration files and by starting the application using the specified script. SRIRAM allows the configuring of the replication to operate in one of two modes.

- Concurrent execution: An instance of the application is launched on all the new replicas.

- Standby execution: New replicas receive a copy of the code, data, and software needed by the machine. The replicas exchange periodic keep-alive messages with the original node. An instance of the application is only launched on the replica when the original node goes down.

Furthermore, if a data-synchronization script is specified in the standard configuration, the script is executed by each of the replicas in order to obtain the latest data changes from the original application node in both of these modes.

If a node with replicas in the system leaves and then rejoins the network, it searches for the existing replicas in operation by floating a query on the spanning tree. The replicas that it discovers cooperate with the node in order to synchronize the code and data. The synchronization process can also be performed at periodic intervals, as determined by the originating node.

The SRIRAM architecture can be used to support automated replication of many different types of applications. Following are some of the common ones.

- *File and video servers*: It is a common practice to use multiple mirrored sites for providing scalable and resilient HTTP (HyperText Transfer Protocol) service, FTP (file transfer protocol) service, and other services with relatively static content. An instance of an application running on a machine, along with its associated data content, can automatically be replicated to other nodes in SRIRAM.
- *Web services*: Web services[11] use the common paradigm of exporting a WSDL (Web Services Description Language) interface for the operations that they support. For most common Web services, a description of the code components (Java classes, beans, etc.) needed for running the Web service is also required. Such a description, and specific software requirements (e.g., a Tomcat server,[12] or the need for JDK** 1.3.1 operating environment) can be advertised and replicated. The presence of replicas can be documented by the replicas themselves in the UDDI directory, which provides a catalog of services.
- *Database applications*: Applications that make heavy use of dynamically changing data in large databases are hard to replicate due to the overheads associated with synchronizing distributed state. For such applications, the replication process would primarily consist of creating hot standbys that can take over in the case of primary system failure. The application software and database replication process can be created automatically on the replicated sites. The replicas will only become operational in the case of failure.

Related work

Earlier work on overlay broadcast and multicast architectures covered a number of approaches, including centralized directory servers,[13–15] flooding-based solutions[2,16] that are typically inefficient, slow distributed spanning tree formation,[17] or requiring voluminous state information in each node.[18] Other work has focused on efficient lookup mechanisms.[19–21] These, however, require exact identifiers for their lookup algorithms, which cannot handle the rich queries (e.g., queries using wildcards) desired. Work on application-layer multicast (e.g., References 6, 22, 23) was primarily directed toward building and maintaining efficient overlay meshes, without considerations of application replication and anonymity. The spanning tree algorithm in Reference 6 does not account for the network and other resources available at each node in the tree construction. While the approach in Reference 22 improves the tree construction process of Reference 6 by first considering a mesh formation and then constructing the tree, it still does not provide a spanning tree in which more powerful nodes are placed higher up in the tree. Also, the algorithm requires significant state management at each node for constructing the spanning tree. Our approach, in contrast, is self-managing, with limited reliance on fixed well-known servers, requires a moderate amount of state information at each node, provides fast and efficient operation, and explicitly includes support for availability, security, and anonymity.

Conclusion and future steps

In this paper, we have described SRIRAM, a system that automates the process by which applications can be replicated in a distributed environment. Any application whose availability is improved by the presence of replicas can benefit from such an automated mechanism. This would include applications that are based on relatively static (or slowly changing) data and do not require very stringent synchronization of the data that they use. The bulk of applications that are available today over the Web, including applications for personalization and transformation of content, fall into this category. The applications that do not benefit from replications are those that operate on highly volatile data and require strict syn-

chronization among the operation of replicas, or use such voluminous data that automated replication becomes too inefficient.

There are several issues that need to be addressed for improving the efficacy of the architecture. First, we are evaluating several alternatives for maintaining hints at the hint-servers. Our goal is to design better hint allocation schemes in which few hints become obsolete and which result in better spanning tree formation. Second, we are addressing the performance issues in the context of a node joining or leaving the mesh: the overhead in updating the tree and accounting for dynamic membership of nodes. Third, we are exploring a better replication scheme which is not purely push-based, but rather a hybrid scheme that combines it with pull-based replication in order to reduce the overhead.

We are currently in the process of implementing a prototype of this architecture and refining the architecture so that it can provide enhanced functions in the future. Some of the functions that we want to incorporate in an extended architecture include the means for authentication of the software running SRIRAM, schemes to bypass portions of the spanning tree in a search, and methods to support multiple spanning trees with different roots. We are also looking at other applications of the SRIRAM architecture, such as the development of highly scalable directory services.

**Trademark or registered trademark of Sun Microsystems, Inc.

Cited references

1. F. Dabek et al., "Building Peer-to-Peer Systems with Chord," *Proceedings of the Eighth IEEE Workshop on Hot Topics in Operating Systems* (HotOS-VIII), IEEE, New York (2001).
2. The Gnutella Protocol Specification, Lime Wire LLC, http://www9.limewire.com/developer/gnutella_protocol_04.pdf.
3. Akamai Technologies, Inc., FreeFlow content distribution service, http://www.akamai.com.
4. Mirror Image® Internet, Inc., *insta*Content® Global Distribution Services, http://www.mirrorimage.net/services/contentdistribution.html.
5. I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," http://www.globus.org/research/papers/ogsa.pdf.
6. P. Francis, Your Own Internet Distribution, http://www.aciri.org/yoid/.
7. C. Ellison et al., "SPKI Certificate Theory," Internet Engineering Task Force, RFC 2693, September 1999, http://www.ietf.org/rfc/rfc2693.txt.
8. T. Dierks and C. Allen, "The TLS Protocol Version 1.0," Internet Engineering Task Force, RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt.
9. S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," Internet Engineering Task Force, RFC 2408, November 1998, http://www.ietf.org/rfc/rfc2408.txt.
10. G. A. Marin, C. P. Immanuel, P. F. Chimento, and I. S. Gopal, "Overview of the NBBS Architecture," *IBM Systems Journal* **34**, No. 4, 564–589 (1995).
11. E. Cerami, *Web Services Essentials*, O'Reilly & Associates, Sebastopol, CA (February 2002).
12. See http://jakarta.apache.org/tomcat/.
13. Napster protocol specification, http://opennap.sourceforge.net/napster.txt.
14. UDDI project, *The UDDI Technical White Paper*, http://www.uddi.org/.
15. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "Almi: An Application Level Multicast Infrastructure," *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems* (USITS), USENIX, Berkeley, CA (2001), pp. 49–60.
16. I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, June 2000, http://freenet.sourceforge.net.
17. S. Radhakrishnan, G. Racherla, C. N. Sekharan, N. S. V. Rao, and S. G. Batsell, "DST—A Routing Protocol for Ad Hoc Networks Using Distributed Spanning Trees," IEEE Wireless Communications and Networking Conference, September 1999, IEEE, New York (1999).
18. K. Psounis, "Active Networks: Applications, Security, Safety, and Architectures," *IEEE Communications Surveys* **2**, No. 1 (First Quarter 1999).
19. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proceedings of ACM SIGCOMM*, August 2001, San Diego, CA, ACM, New York (2001).
20. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proceedings of ACM SIGCOMM*, August 2001, San Diego, CA, ACM, New York (2001).
21. I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, *Lecture Notes in Computer Science*, Vol. 2009, H. Federrath, Editor, Springer, New York (2001), pp. 46–66.
22. Y. Chu, S. Rao, and H. Zhang, "A Case for End-System Multicast," *Proceedings of ACM SIGMETRICS*, July 2000, Santa Clara, CA, ACM, New York (2000).
23. Y. Chawathe, S. McCanne, and E. A. Brewer, "*An Architecture for Internet Content Distribution as an Infrastructure Service*," February 2000, unpublished work.

Accepted for publication August 29, 2002.

Dinesh C. Verma *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: dverma@us.ibm.com).* Dr. Verma received a B.Tech. degree in computer science from the Indian Institute of Technology, Kanpur, India, in 1987 and a Ph.D. degree in computer science from the University of California, Berkeley, in 1992. Since then he has worked at the IBM Thomas J. Watson Research Center and Philips Research Laboratories. He is currently a research manager at the IBM Research Center and oversees research in the area of edge networking. His current research interests include content distribution networks, policy-based networking, and performance management in networked systems.

Sambit Sahu *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: sambits@us.ibm.com).* Dr. Sahu is a research staff member in the Networking Software and Services group in IBM Research. He received his Ph.D. degree in 2001 from the University of Massachusetts, Department of Computer Science. Since joining IBM, his research has focused on overlay-based communication, content distribution architecture, and design and analysis of high-performance network communication protocols. He has published a number of papers on differentiated services, multimedia, and peer-to-peer communication.

Seraphin Calo *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: scalo@us.ibm.com).* Dr. Calo is a research staff member at IBM Research. He received the M.S., M.A., and Ph.D. degrees in electrical engineering from Princeton University, Princeton, New Jersey. He has worked, published, and managed research projects in a number of technical areas, including: queuing theory, data communications networks, multiaccess protocols, expert systems, and complex systems management. He has been very active in international conferences, particularly in the systems management area. His current research interests include: content distribution networks, distributed applications, services management, and policy-based computing.

Anees Shaikh *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: aashaikh@watson.ibm.com).* Dr. Shaikh is a research staff member in the Networking Software and Services group in IBM Research. He received his Ph.D. degree in 1999 from the University of Michigan, Department of Computer Science and Engineering. Since joining IBM, his research has focused on Internet services infrastructure, particularly the areas of wide-area load balancing, performance measurement of Web-based applications, and content distribution architecture. He has published a number of papers on load-sensitive routing, middleware for real-time communication, and multicast routing.

Isabella Chang received her B.S. degree in 1986 and the M.S. degree in optics in 1986, both from the University of Rochester. From 1998 to 2002, she worked at the IBM Thomas J. Watson Research Center on topics related to computer communication networks with focus on implementation of network control software, such as IP quality-of-service Resource Reservation Protocols (RSVP), and DSML (Directory Services Markup Language) gateways.

Arup Acharya *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: arup@us.ibm.com).* Dr. Acharya is a research staff member in the Edge Networking group at the IBM Thomas J. Watson Research Center. His research interests include emerging network architectures such as VoIP, MPLS, and IPv6, as well as mobile wireless networking. Before joining IBM, he was with NEC C&C Research Laboratories in Princeton, New Jersey, between May 1995 and November 1999. He received a B.Tech. degree in computer science from the Indian Institute of Technology, Kharagpur, and a Ph.D. degree from Rutgers University in 1995. He has published numerous papers in his areas of interest and has been a program committee member of leading technical conferences in mobile networking.