# Load Sharing in Java RMI Middleware for Active Networks*

Cheng-Zen Yang, Chia-Lung Chen, and I-Hsuan Huang
Department of Computer Science and Engineering
Yuan Ze University
Chungli, Taiwan, R.O.C.
E-mail: {czyang,jlchen,ihhuang}@syslab.cse.yzu.edu.tw

***Abstract*— In our past research, ActiveRMI was proposed to employ the native benefits in active networks to improve system performance. However, in the past ActiveRMI design, the load sharing issue has not been considered completely. Although ActiveRMI shows its prominence to the traditional Java RMI programming paradigm, the front edge active routers may still become the performance bottleneck if an extremely large number of client requests burst into these routers. Therefore, in this paper we propose a dynamic server-initiated distributed load sharing scheme for ActiveRMI. The proposed load sharing scheme has two major design features. First, the average user response time can be reduced by sharing loads of overloaded active routers with other nearby active routers which are lightly loaded or moderately loaded. Second, the overhead incurred by load sharing transfer is kept minimal by initiating the transfer on demand. To study the performance improvements, we have conducted preliminary simulation experiments. The results show that the performance of ActiveRMI with load sharing support is superior to the performance of ActiveRMI without load sharing support.**

***Keywords*— Load sharing, ActiveRMI, Java RMI, Active Networks.**

## I. Introduction

Since 1994, the concept of active networks [14, 13] has been proposed to introduce the programmability to traditional routers. The programmability enhancement mainly copes with three problems existing in the traditional network environment: (1) the difficulty for new technology integration, (2) innate poor performance due to the protocol stack structure, and (3) the difficulty of the deployment of new services. The main reason behind these problems is that traditional routers are just responsible for data transmission. On the contrary, the routers, called *active routers* or *active nodes*, in active networks are designed to be able to execute the user-specific code embedded in the specialized network packets, called *active packets* or *capsules*. Therefore, active networks can be used to enhance the performance of traditional network services, and facilitate the deployment of new network services. In many studies (e.g. [3, 7]), the prominent flexibility and programmability of active networks have been justified.

In the research and development of active networks, one major absence is the lack of the discussion of a suitable programming paradigm for the programmable active routers. For example, traditional RMI/RPC middleware design is not aware of the programmability introduced in active networks. The applications developed with the traditional RMI/RPC paradigm cannot be benefitted in active networks. By contrast, their performance may be hindered because the active routers have the overhead to process active packets. In addition, the system scalability of the traditional RMI/RPC paradigm on active networks cannot be effectively improved due to its employment of a remote centralized server.

To provide a suitable programming paradigm for active networks with Java middleware, Wueng and Yang [19, 20] proposed a new RMI programming paradigm called ActiveRMI that can exploit the feature of dynamic code execution and deployment in active networks. In ActiveRMI, when a client invokes a service, its requests are intercepted and processed on a nearby active routers rather than the remote RMI server. The interceptive active router then acts as an service agent to download the service code from the remote RMI server and initiate the service to satisfy the client requests. In addition, the interceptive active router maintains a service code cache for future client invocations. In this new service model, ActiveRMI achieves four major advantages. First, it shortens the service response time by performing remote services at the proximate active routers. Second, the workloads of remote RMI servers are shared with the intermediate active routers in this implicit multi-tier architecture. The central bottleneck problem in traditional tier-to-tier RMI service model is relieved. Third, service availability is highly improved because remote services are distributed at nearby active routers. The interceptive active routers become the agents of the remote RMI server to satisfy client requests. Last, the system scalability is also highly improved because most client requests are satisfied at the nearby active routers. From their preliminary experimental results, ActiveRMI shows its prominence in distributed computing benchmarks [19, 20, 21].

As shown in the preliminary experiments, ActiveRMI effectively reduces the network access time and shortens the user response time. However, although ActiveRMI can relieve the heavy load of the remote RMI server by distributing the workload to the active routers nearby the clients, the active routers in the original ActiveRMI design may still become performance bottlenecks if many client requests burst into the routers. In such a situation, while the workload of the remote server is reduced, the workload of each active router is increased substantially. Observing this situation, we are motivated to conduct research on the load distribution issue to prevent some active router from being the performance

*This research was supported by National Science Council of R.O.C. under grant NSC 93-2213-E-155-046.

bottleneck.

Load distribution has been studied for many years (e.g [12, 10, 8, 23, 22, 2, 6]). According to the criteria mentioned in [12], it can be further discussed as load sharing and load balancing. The major concern of load balancing is how to equalize the loads of all sites. By contrast, the major concern of load sharing is how to maximize the system performance by relieving the system peak overload with other available sites. Since load balancing algorithms require higher network bandwidth to actively balance server loads, load sharing approaches are more suitable for active networks.

Previous load sharing approaches can be further classified from different viewpoints, such as *centralized* vs. *distributed*, *static* vs. *dynamic*, and *server-side* and *client-side* [5, 12, 16]. From the viewpoint of the location of the load dispatcher, they can be classified into two categories for our discussion [12]: the centralized approaches (e.g. [2, 4, 6]) and the distributed approaches (e.g. [10, 8, 23, 22]). In the centralized approaches, there is a central host or agent to make the load migration decision. For example, in the Comet algorithm a central host decides whether an agent should be migrated [6]. In such a centralized system, two important issues should be considered. First, to prevent the central host from being the serious system bottleneck, the tasks executed on the central host need to be lightweighted. Second, to prevent the central host from being the single point of failure, fault tolerance mechanisms need to be incorporated in the system, which may result in performance degradation. For an active network environment, centralized approaches are not suitable because it is hard to designate an active router as the central agent across several administrative boundaries. Furthermore, the communication lag between active routers may be too large for collecting up-to-date load information.

In contrast, no central host exists for the distributed load balancing approaches. For example, the CAPE system provides a peer-to-peer protocol to distribute load information between load balancing objects [22]. For an active network environment, a distributed load balancing approach is more suitable for its distributed collaboration nature. However, two important issues should be considered. First, the communication overhead of distributively collecting load information needs to be minimized as much as possible. Otherwise, the system scalability is limited to a small range. Second, the demand-driven policy for deciding which is the candidate to initiate load sharing influences the system performance. Basically, there are three kinds of demand-driven policies [12, 1]: sender-initiated policies, receiver-initiated policies, and symmetrically initiated policies. Since usually active routers are not tightly located in a local network environment, the sender-initiated policies are more suitable because other two kinds of policies may incur a lot amount of network transfer.

In this paper, we propose a dynamic server-initiated distributed load sharing scheme for ActiveRMI with two major design goals. First, the average user response time can be reduced by sharing loads of overloaded active routers with other nearby active routers which are lightly loaded or moderately loaded. Second, the overhead incurred by load sharing transfer is kept minimal by initiating the transfer on demand.

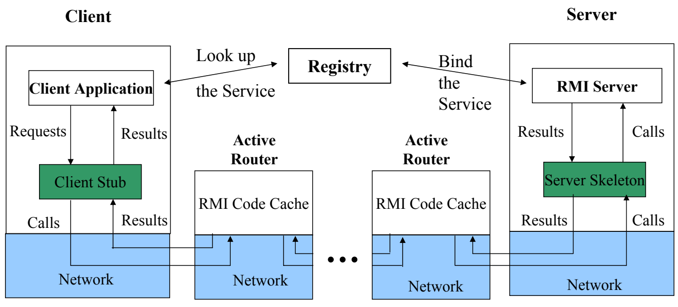

Fig. 1. The ActiveRMI infrastructure.

To study its performance improvement, we have conducted preliminary simulation experiments. The results show that the performance of ActiveRMI with load sharing support is superior to the performance of ActiveRMI without load sharing support.

The rest of the paper is organized as follows. Section 2 introduces the related work including the recent progress in ActiveRMI and past studies on load sharing and balancing. Section 3 describes the load sharing design and its functionalities. Section 4 presents the results of simulation experiments. Finally, Section 5 concludes the paper and discusses the future research directions.

## II. Related Work

In this section, we first briefly review the programming model in ActiveRMI [19, 20, 21] which can automatically migrate remote services to the nearby active routers. Then we briefly review several recent studies on distributed load sharing and balancing, and discuss their features with our proposed load sharing scheme.

### A. *ActiveRMI*

In the ActiveRMI paradigm, each active router is designed to be aware of Java RMI service invocations, and to act as an agent of the remote RMI servers by executing the service code from its local code cache. When the active router gets client requests, it first checks the local code cache. If there is a valid copy of the service code, the router immediately activates the service to satisfy the client requests. Otherwise, it tries to get the service code from the remote RMI server by forwarding the client requests. Figure 1 illustrates the ActiveRMI infrastructure.

To facilitate such an execution model, the client stub is responsible for handling the underlying packet transmissions on the client side. The traditional RMI code generator is also re-designed to generate ActiveRMI client/server stubs. Figure 2 illustrate the procedure of code generation in which programmers provide interface files and Java program code to ARMIGen. ARMIGen translates these files to produce ActiveRMI executable binary files. Since ActiveRMI is developed on Janos [11, 15] and ANTS [17, 18], it implements an extended ANTS protocol to practice code caching.

ANTS is designed at MIT as an active network platform with two distinctive design features [17, 18]. First, ANTS supports customized code execution to provide various network services. Second, ANTS supports the establishment and

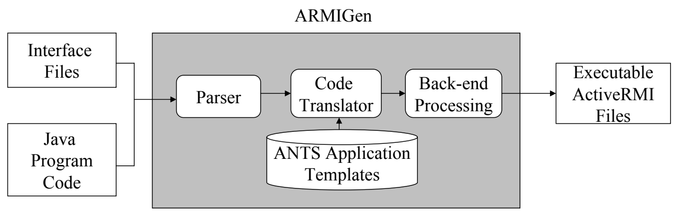

Fig. 2. Code generation with ARMIGen.

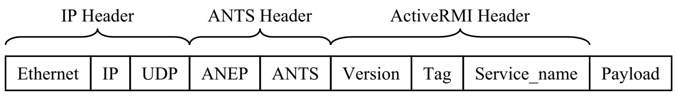

Fig. 3. Format of an ActiveRMI capsule.

deployment of new protocols. In ANTS, capsules carry user code and data to achieve these design features. A hierarchical capsule model is specified to support various application requirements. When a capsule arrives at an active router, the active router checks the protocol of this capsule. If the router does not have the protocol code, it sends a code-downloading request back to the previous active router. The capsules are then kept provisional until the required code is prepared. If all required code segments are prepared, the capsules are awakened and ANTS continues the execution.

In ANTS, capsules and protocols are defined in Java files. An ANTS application consists of the following files: a main program, definitions of capsules and protocols, and script files. The script files contain a configuration table, a routing table, and a start file that is used to activate the service. The configuration table has the following information: the physical IP address, the logical IP address for a specified ANTS application, and the network port. The Janos virtual machine receives capsules from the fixed network ports and dispatches these capsules to different domains according to the logical IP information.

The routing table records the routing information: the physical IP addresses, the logical IP addresses, and the network ports of active routers. Active routers behave as the tunnels because traditional routers may exist in current IP networks. Due to the limitation of the current ANTS experimental platform, the position of the active routers need to be explicitly specified in the routing table. As specified in the routing table, the capsules are transferred through the designated active routers to the active routers.

ActiveRMI is designed to utilize the programmability benefits of active networks to improve the performance of Java RMI invocation with little extra programming efforts. Each active router has an ActiveRMI code cache to keep RMI service code. With the RMI code caches, the users repose time is reduced and the scalability is enhanced because the RMI services cached on the intermediate active routers can satisfy the requests as soon as possible. In the ActiveRMI code caching mechanism, a manager called ARMIMan maintains the RMI service code execution and the code cache.

ActiveRMI applications use several capsules to cooperate with ARMIMan and the registry: `lookup`, `registry`, `argument`, and `result`. Figure 3 shows the format of an ActiveRMI capsule. Since the underlying platform is ANTS, the format of the ActiveRMI capsules follows the ANTS capsule format. An RMI server uses `registry` capsules to bind RMI service information to the registry. The client sends `lookup` capsules to the registry to search for the RMI services. The registry processes the `lookup` capsules and then replies to the client with the registration information and the corresponding stub. The client can then invoke the stub to send `argument` capsules to the RMI server. When ARMIMan detects `argument` capsules, it intercepts these capsules and executes the RMI service code from the local RMI code caches. When the request is satisfied, the client stub gets the `result` capsules and replies the result to the client applications.

ActiveRMI achieves three improvements. First, user response time is reduced in ActiveRMI compared with the traditional Java RMI. Second, the server workload is shared with the active routers. The centralized bottleneck problem is alleviated. Third, the amount of network traffic between the clients and the RMI server is reduced because the active routers act as the agents to intercept client requests and execute the cached services immediately. More details can be found in [19, 20, 21].

### *B. Related Load Sharing and Balancing Schemes*

Even after several decades, distributed load sharing and balancing is still a growing research area for distributed computing (e.g [8, 9, 23, 22]). In active networks, studies on distributed load sharing and balancing have been also conducted to employ the advantages embedded in programmability. In 2000, Yoshihara et al. proposed a dynamic load balancing scheme for distributed management in active networks [23]. To perform load balancing on remote servers, management scripts are dynamically downloaded to and executed on the active routers according to the average CPU utilization of the system and the needed network bandwidth. However, in their distributed management paradigm, they do not consider how to migrate the remote services to the nearby active routers. In addition, they focus on the load balancing of distributed management rather than remote services. Since the management system does not directly respond the client requests, the hot-spot phenomenon of request burst rarely occurs as a serious problem. By contrast, ActiveRMI considers service migration in the native design. ActiveRMI needs to directly handle the request burst problem with a more request-centric load sharing scheme.

In Active Anycast, Miura et al. proposed a mechanism to dynamically dispatch client requests to a lightly loaded remote server with active routers [9]. The client sends the requests as the original Anycast design. The active routers on the transfer path change the destination address according to the collected load information. Therefore, Active Anycast achieves a good performance of load balancing. From their simulation results, Active Anycast shows significant improvements with only a small number of active routers in a WAN environment.

From the viewpoint mentioned in [12], the approach incorporated in Active Anycast is a load sharing scheme. However,

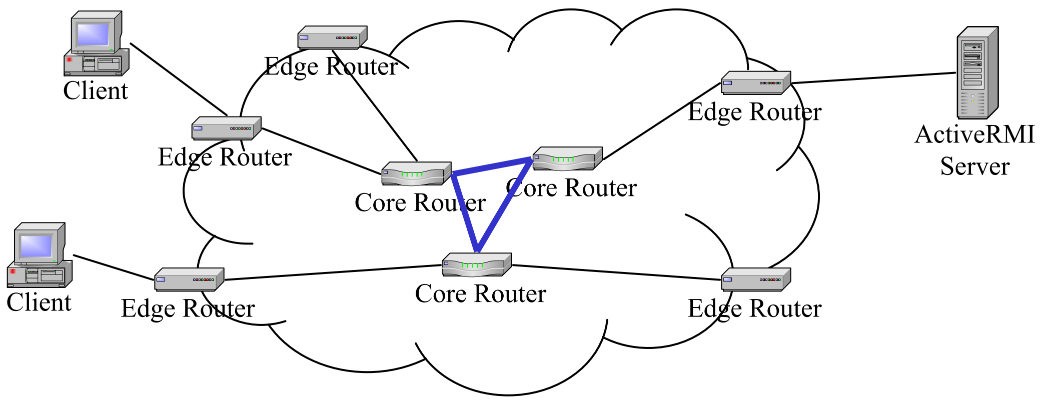

Fig. 4. Example of an active network environment in which all routers are active routers.

in Active Anycast, the native benefits of the programmability and the capability of deploying new services in active networks are not incorporated in the system design. The loads are shared between remote replicated servers, not active routers. Since the hop count of the communication path may be large, the average user response time could be still lengthy if some intermediate active routers are heavily loaded. By contrast, ActiveRMI exploits the native benefits of the programmability and the capability of deploying new services in active networks to migrate remote services to the nearby active routers. The length of the communication path between a client and the service destination is thus minimized as possible.

In 2001, Yamaguchi and Maruyama proposed a distributed load balancing scheme in the CAPE system [22]. A peer-to-peer (P2P) load balancing protocol takes the responsibility of load balancing. The P2P feature makes load balancing in the CAPE system truly distributed. However, since the load information in this P2P scheme is exchanged with a broadcasting mechanism, the system scale is thus limited to a small range. On the contrary, the load sharing information in our proposed scheme is exchanged in a small group of active routers. Therefore, our proposed scheme can be highly scalable.

## III. The Load Sharing Scheme

The proposed load sharing scheme is a dynamic sender-initiated distributed scheme. Load dispatching depends on the current system state rather than on a predefined sharing scheme such as hashing. Several design issues needed to be discussed in this dynamic sender-initiated scheme, such as the scope of the load dispatching and the selection policy. In the following, we first describe the system architecture to which the proposed load sharing scheme is applied. Then the design considerations are elaborated.

### *A. The System Architecture*

The system architecture discussed in this paper is assumed to be a pure active environment as shown in Figure 4 to clearly elaborate our proposed dynamic sender-initiated distributed load sharing scheme. In this pure environment, all routers are active routers. All clients, active routers, and servers can execute the ActiveRMI protocol. Since traditional routers only perform packet forwarding operations, this assumption does not loose any generality if the environment has some traditional routers on the communication paths.

The active routers are further classified into two categories: the edge active routers and the core active routers. The edge active routers are the routers closest to the clients and the servers. Therefore, they may intercept a large number of client requests. Other active routers are core routers which are the intermediate routers between two edge active routers.

### *B. Distributed Load Sharing*

Several issues need to be discussed in our proposed dynamic sender-initiated distributed scheme, including load indexes, the transfer policy, the information policy, and the location policy. They are elaborated as follows.

*1) Load Indexes and Transfer Policy:* This is the key issue to design a distributed load sharing scheme for active networks. Since the number of execution environments (EE) is dynamically changed as the active router is running, and is an effective approximation for system resource consumption such as the average CPU utilization, the average number of EE queue length over some period, $AVQ_{EE}$, is considered. When $AVQ_{EE}$ of an active router is larger than an administrator-defined threshold $TH_{AVQ}$, the active router (sender) starts the load sharing scheme to try to transfer its load to another lightly loaded active router. If $AVQ_{EE}$ of the destination active router is lower than its $TH_{AVQ}$, the router accepts the sharing request and starts to run the EE. Otherwise, it rejects the the sharing request, and the sender tries to find another possible destination.

To avoid suffering long network latency by selecting a very remote active router as the migration destination, the network transfer latency $L_{trans}$ and an upper bound of the hop count $HOP_{max}$ are also considered. They are used to control the farthest migration boundary to avoid the potential message flooding over the whole networks. The router whose $L_{trans}$ is larger than the threshold $TH_{trans}$ or whose hop count is larger than $HOP_{max}$ will not be considered as a destination candidate. Since $L_{trans}$ is changed as time varies, the boundary will be also dynamically changed as new load information is collected.

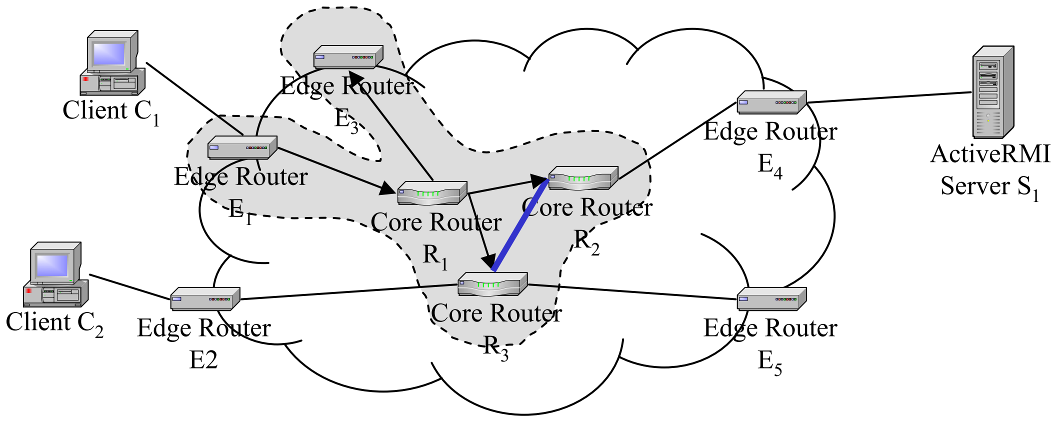

Fig. 5. The group configuration of an edge active router $E_1$ for load sharing when $HOP_{max} = 2$.

Figure 5 shows an example in which we assume that $HOP_{max}$ of the active router $E_1$ is set as two, and the latency to the routers which are inside the hop count boundary is smaller than $E_1$'s $TH_{trans}$. Therefore, the routers $E_3$, $R_1$, $R_2$, and $R_3$ are formed as the possible *sharing group* of $E_1$.

However, the sharing group is not symmetric. The formation of a sharing group totally depends on the configuration of each active router. That is, the sharing group of $E_3$ does not include $E_1$ if $E_3$'s $HOP_{max}$ is one.

*2) Information Policy:* Since the proposed scheme is a distributed scheme, each active router needs to collect the load information of the member nodes in its sharing group. Here a periodical gossip model is used to propagate the load information. In this model, each node just collects information from its neighbor nodes, and maintains a load table to record the load information of the active routers in its sharing group and the sharing groups of its neighbor nodes. At the mean time, it also broadcasts its load information to the neighbor nodes. For example, $R_1$ may have an empty sharing group, but it still records $E_3$'s load information because $E_1$ needs this information. Although the collected information may not be very accurate in this model, this approach avoids message flooding overheads.

*3) Location Policy:* Since the sender has the load information of all active routers, it can create a receiver list in the load-decreasing order. Then the sender shares the service load with the nodes in the order of the receiver list and adjusts the load information to update the receiver list accordingly. If the selected node is overloaded when sharing is initiated (this may happen because there is a time gap to reflect the most up-to-date load information), the next node in the list is selected.

## IV. Simulation Study

To study the performance of the proposed load sharing scheme, we have conducted preliminary simulation experiments. In the experiments, we assumed that all routers were active routers. The context switching time and the EE startup time were negligible. The services had been all cached in each routers. The processing power of each active router was represented by $TH_{AVQ}$. The range of the service workloads was randomly generated between 100 and 1000 time units. The communication latency between clients and the edge routers was 10 time units, and the latency between two routers 1 time unit. In the simulation, the sharing boundary was controlled only by $HOP_{max}$ for the simplicity sake.

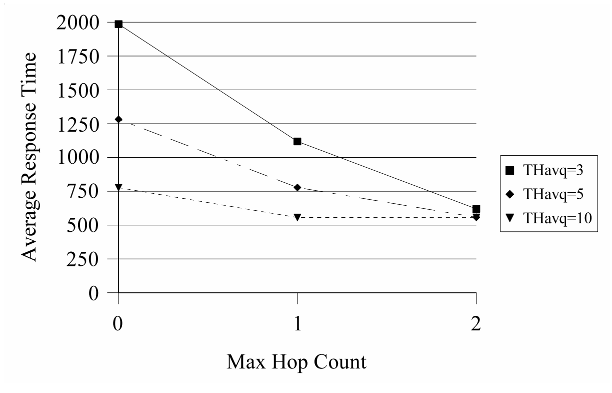

Fig. 6. The experimental simulation results.

Figure 6 shows the simulation results. In the figure, $HOP_{max} = 0$ means no load sharing, and $HOP_{max} = 2$ means the sharing is performed in the range of 2 hops. From the figure, we can notice that the load of the active routers can be effectively shared.

Interestingly, we also notice that the average response time is slightly increased when some service loads are shared with remote active routers. The main reason is that our load index mainly considers the system queue length, but neglects the communication overhead. However, the influence is very small, and can be negligible.

## V. Conclusions and Future Work

Load sharing is a crucial issue in distributed computing. In the past ActiveRMI design, this issue has not been considered completely. Although ActiveRMI shows its prominence to the traditional Java RMI programming paradigm, the front edge active routers may still become the performance bottleneck if a vast number of client requests burst into these routers. Therefore, a load sharing scheme is required to relieve this potential performance bottleneck.

In this paper, we propose a dynamic server-initiated distributed load sharing scheme for ActiveRMI. There are two major design goals. First, the average user response time can be reduced by sharing loads of overloaded active routers with other nearby active routers which are lightly loaded or moderately loaded. Second, the overhead incurred by load sharing transfer is kept minimal by initiating the transfer on demand.

To study the performance improvements, we have conducted simulation experiments. Although the experiment is preliminary, the results show that the performance of ActiveRMI with load sharing support is superior to the performance of ActiveRMI without load sharing support.

To conclude, the proposed load sharing scheme indeed improves the bottleneck problem in the traditional ActiveRMI paradigm. However, many practical implementation issues need to be further discussed. For example, how to piratically define the load information in real systems is one of the challenging implementation problems. In the future, we plan to conduct experiments to study the network performance of ActiveRMI with load sharing support in a large scale environment.

## REFERENCES

[1] M. Arora, S. K. Das, and R. Biswas, "A De-centralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments," in *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)*, Aug. 2002, pp. 499–505.

[2] J. Aweya, M. Ouellette, D. Y. Montuno, B. Doray, and K. Felske, "An Adaptive Load Balancing Scheme for Web Servers," *International Journal of Network Management*, vol. 12, pp. 3–39, 2002.

[3] K. L. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz, "Directions in Active Networks," *IEEE Communications Magazine*, vol. 36, no. 10, pp. 72–78, Oct. 1998.

[4] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu, "The State of the Art in Locally Distributed Web-Server Systems," *ACM Computing Surveys*, vol. 34, no. 2, pp. 263–311, June 2002.

[5] T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141–154, Feb. 1988.

[6] K.-P. Chow and Y.-K. Kwok, "On Load Balancing for Distributed Multiagent Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 8, pp. 787–801, Aug. 2002.

[7] U. Legedza, D. J. Wetherall, and J. Guttag, "Improving the Performance of Distributed Applications Using Active Networks," in *Proceedings of the 7th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM'98*. San Francisco, CA: IEEE, Apr. 1998, pp. Vol. 2, 590–599.

[8] W. Leinberger, G. Karypis, V. Kumar, and R. Biswas, "Load Balancing Across Near-Homogeneous Multi-Resource Servers," in *Proceedings of the Heterogeneous Computing Workshop (HCW)*, 2000, pp. 60–71.

[9] H. Miura, M. Yamamoto, K. Nishimura, and H. Ikeda, "Server Load Balancing with Network Support: Active Anycast," in *Proceedings of the Second International Working Conference Active Networks (IWAN 2000)*, Tokyo, Japan, Oct. 2000, pp. 371–384.

[10] C. Park and J. G. Kuhl, "A Fuzzy-Based Distributed Load balancing Algorithm for Large Distributed Systems," in *Proceeding of the Second International Symposium on Autonomous Decentralized Systems (ISADS)*, 1995, pp. 266–273.

[11] L. Peterson, Y. Gottlieb, M. Hibler, P. Tullmann, J. Lepreau, S. Schwab, H. Dandekar, A. Purtell, and J. Hartman, "An OS Interface for Active Routers," *IEEE Journal on Selected Areas of Communication*, vol. 19, no. 3, pp. 473–487, Mar. 2001.

[12] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *IEEE Computer*, vol. 25, no. 12, pp. 33–44, Dec. 1992.

[13] D. L. Tennenhouse, J. Smith, W. D. Sincoskie, D. J. Wetherall, and G. Minden, "A Survey of Active Network Research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

[14] D. L. Tennenhouse and D. J. Wetherall, "Towards an Active Network Architecture," in *Proceedings of Multimedia Computing and Networking (MMCN 96)*, San Jose, CA, Jan. 1996.

[15] P. Tullmann, M. Hibler, and J. Lepreau, "Janos: A Java-Oriented OS for Active Network Nodes," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 3, pp. 501–510, Mar. 2001.

[16] J. Wei, C.-Z. Xu, and X. Zhou, *Internet Encyclopedia*. John Wiley & Sons Publisher, 2004, vol. 2, ch. Load Balancing on the Load Balancing on the Internet, pp. 499–514.

[17] D. J. Wetherall, "Service Introduction in an Active Network," Ph.D. dissertation, Massachusetts Institute of Technology, Feb. 1999.

[18] D. J. Wetherall, J. Guttag, and D. L. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," in *Proceedings of the First IEEE Conference on Open Architectures and Network Programming (OPENARCH'98)*, Apr. 1998, pp. 117–129.

[19] M.-C. Wueng and C.-Z. Yang, "Design of Consistency Maintenance in ActiveRMI Code Caching," in *Proceedings of 2003 Active Networking Workshop*, Chungli, Taiwan, Sept. 2003, pp. 17–22.

[20] ——, "Design and Implementation of a Java RMI Caching Mechanism on Active Networks," in *Proceedings of the 6th International Conference on Advanced Communication Technology (ICACT 2004)*. Phoenix Park, Korea: IEEE, Feb. 2004, pp. 561–566.

[21] M.-C. Wueng, F.-F. Yang, and C.-Z. Yang, "A Novel Java RMI Middleware Design for Active Networks," in *Proceedings of IEEE TENCON 2004*. Chiang Mai, Thailand: IEEE, Nov. 2004, pp. C068–C071.

[22] S. Yamaguchi and K. Maruyama, "Autonomous Load Balance System for Distributed Servers Using Active Objects," in *Proceedings of the 12th International Workshop on Database and Expert Systems Applications (DEXA 2001)*, Munich, Sept. 2001, pp. 167–171.

[23] K. Yoshihara, K. Sugiyama, K. Nakao, T. Oda, and S. Obana, "Dynamic Load-Balancing for Distribute management in Active Networks," in *Proceedings of the Second International Working Conference Active Networks (IWAN 2000)*, Tokyo, Japan, Oct. 2000, pp. 326–342.