Dynamic Load-Balancing of Jini and .NET Services *

Ying Chen Lin Sy-Yuan Li Yuan-Shin Hwang
Department of Computer Science and Engineering
National Taiwan Ocean University
Keelung 20224
Taiwan

Abstract

Jini and .NET are the two most popular distributed computing environments nowadays. Jini architecture provides an infrastructure for defining, advertising, and finding services in a network, while .NET lets developers build Internet-based, distributed applications using .NET Remoting. These two environments are very similar in many aspects and both are capable of facilitating implementation of distributed programs. However, they are not compatible. Clients in Jini can not request remote services from .NET systems and clients in .NET can not find Jini services. Furthermore, none of these environments provides load-balancing mechanisms. To solve these two problems, this paper integrates Jini and .NET and proposes a smart proxy architecture. Consequently, Jini and .NET services can now make themselves visible to both Jini and .NET clients, and any Jini or .NET client can transparently switch to a less-loaded Jini or .NET service through a smart proxy once the current service is too busy. Specifically, this paper solves two important issues for Jini and .NET: interoperability and dynamic load-balancing. Experimental results show that this technique can dynamically distribute Jini and .NET clients quite evenly over Jini and .NET services when proper load-balancing strategies are implemented onto smart proxies.

1. Introduction

Jini is a Java-based infrastructure developed by Sun Microsystems that can provide all the services necessary to support parallel and distributed applications [1, 6, 19]. Since its introduction, Jini has been applied to enterprise applications [11], grid computing [2], embedded systems [3], sensor networks [18], and even home networking [9]. A Jini service can place its own proxy object in any Lookup services in order to register to offer itself for use to Jini clients. Any client looking for a service finds Lookup services and receives a service proxy from any available Lookup services. Afterward, the client can request the service directly through the service proxy.

The .NET framework is Microsoft's initiative to deliver a new way of building and deploying applications and services [8]. Specifically, .NET remoting provides powerful remote interaction among objects and hence enables developers to build widely distributed applications easily, whether application components are all on one computer or spread out across the entire world [14]. .NET remoting provides an abstract approach to interprocess communication that separates remote objects from a specific client or server application domain and from a specific mechanism of communication. As a result, it is flexible and easily customizable.

Jini and .NET are the two most popular distributed computing environments nowadays. Although these technologies are implemented quite differently and are based on different philosophies, they are remarkably similar in many ways. However, they are not compatible. In other words, clients in Jini can not request remote services from .NET systems and .NET clients can not find Jini services. This paper presents an approach to make Jini and .NET interoperable. Specifically, Jini and .NET services can now make themselves visible to both Jini and .NET clients by registering to the Jini Lookup services, while Jini or .NET clients can locate Jini or .NET services through the Lookup services and then make requests directly to Jini or .NET services. Furthermore, since requests to services are in fact initiated through proxies, clients will not be able to tell whether they are connected to Jini services or .NET services.

In addition to the issue of interoperability for Jini and .NET, another important topic is dynamic load-balancing among Jini and .NET services. Since the prototype implementations of Jini and .NET do not provide any load-

*This research was supported by NSC grant NSC94-2213-E-019-008 and MOEA project 95-EC-17-A-01-S1-034

balancing mechanism [15, 17], programmers have to explicitly incorporate code in clients to communicate with load-balancing systems. Couples of load-balancing systems have been proposed and analyzed, and clients in these systems make requests to the load-balancing systems [4, 5, 7]. In other words, the process of load balancing is not transparent to clients. As a result, the clients in these systems will look significantly different from the clients in the original Jini system. This paper proposes a smart proxy architecture as a remedy [10, 12, 13]. A Jini or .NET client can easily switch to any less-loaded Jini or .NET service through a smart proxy without knowing it.

In order for a smart proxy to decide if a client should switch from the current service to a less-loaded Jini or .NET service, the load information of services is stored in the ServiceTable on Jini Lookup services. A smart proxy will consult the ServiceTable to choose an appropriate Jini or .NET service for the client according to its load-balancing strategy. As a result, the switching process is transparent to the client. Several strategies have been developed and the experimental results demonstrate these strategies can distribute loads evenly.

Since smart proxies of clients consult the ServiceTable on Lookup services before making requests, services must periodically report their load information to the Lookup services. However, such update actions will introduce extra network communications. This paper presents an easy way to reduce such extra overheads—a service updates its load information when it renews its lease with Lookup services.

The main results of this paper are as follows:

- This paper integrates Jini and .NET. Jini and .NET services can now make themselves visible to both Jini and .NET clients by registering to the Jini Lookup services, while Jini or .NET clients can locate Jini or .NET services through the Lookup services and then make requests directly to Jini or .NET services.
- This paper presents a dynamic load-balancing mechanism for Jini and .NET, which is based on the smart proxy architecture. A Jini or .NET client can easily switch to any less-loaded Jini or .NET service when the current service is over-loaded.
- The load information of services is stored in Lookup services, and is updated when services renew leases with Lookup services. Consequently, updating load information does not incur additional network communications.

The rest of this paper is organized as follows. Section 2 briefly overviews main features of the Jini and .NET, and Section 3 introduces the smart proxy architecture for dynamic load-balancing. Experimental results will be presented in Section 4, and Section 5 concludes this paper.

2. Background

This section provides brief descriptions of the Jini and .NET.

2.1. Jini

Jini is a Java-based connection technique developed by Sun Microsystems that can be used to build a flexible network of resources and services to be shared by a group of clients [1, 6, 19]. It provides the necessary protocols for services to register themselves with Lookup services and for clients to discover services.

2.1.1. Jini Proxies

Proxies play an important role in the Jini system. They act as the gateway to remote services. That is, they deal with any network-related functions for clients, transmitting any parameters to remote services and receiving any return values from the services.

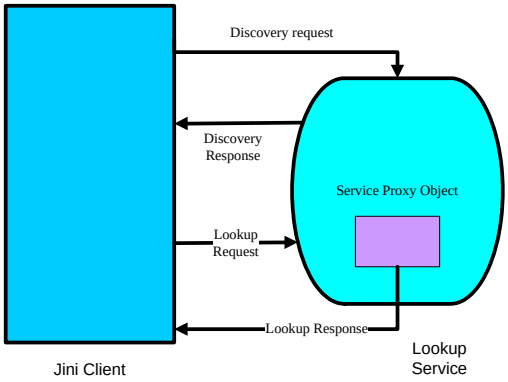


Figure 1. Service Registration

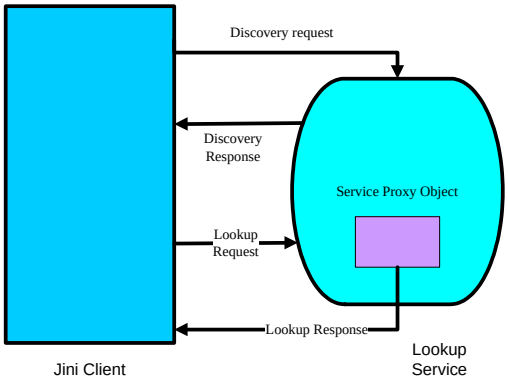


Figure 2. Service Lookup

A Jini service can place its own proxy object in any Lookup services in order to register to offer itself for use

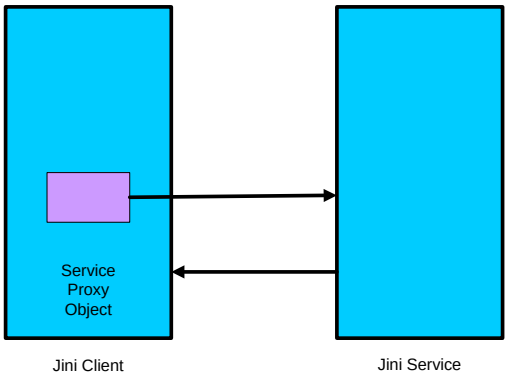


Figure 3. Service Request

to Jini clients, as shown in Figure 1. Any client looking for a service finds Lookup services and receives a service proxy from any available Lookup services (Figure 2). Afterward, the client can request the service directly through the service proxy, as depicted in Figure 3.

2.1.2. Leasing

Jini deploys the notion of leasing to allow clients and services to leave easily without disrupting other members. Jini's leasing model sets time limits that services are available to clients. A Jini member offering a resource does so through a lease, which represents a period of time which the service is available. As a result, a service usually periodically renews its lease with the Lookup service.

2.2. .NET

.NET is a set of Microsoft software technologies for connecting information, people, systems, and devices [15].

2.2.1. C#

C# is a new programming language designed for building a wide range of enterprise applications that run on the .NET Framework. It is simple, modern, type safe, and object oriented. C# is very similar to Java syntactically and can be interoperable with Java via bridging tools such as JNBridge.

2.2.2. .NET Remoting

Microsoft .NET remoting provides a framework that allows objects to interact with one another across application domains [14]. The framework provides a number of services, including activation and lifetime support, as well as communication channels responsible for transporting messages to and from remote applications. .NET Remoting has a logical and cohesive object model that facilitates both simple configuration changes and advanced extensions to the .NET

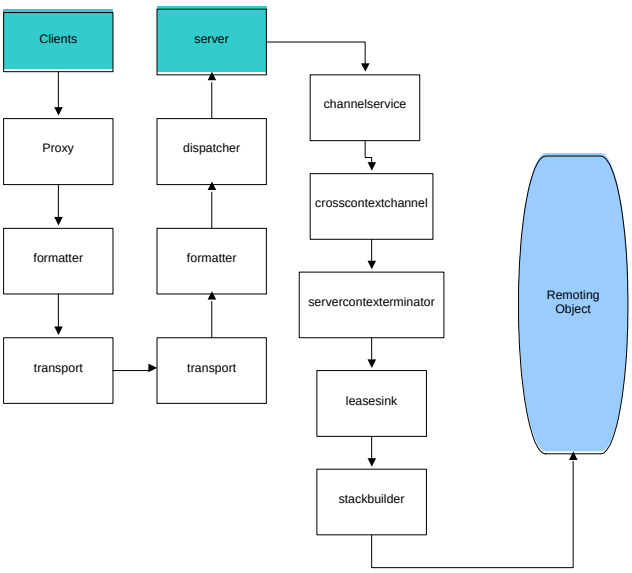


Figure 4. .NET Remoting Infrastructure

Remoting infrastructure. .NET Remoting architecture is shown in Figure 4.

2.2.3. Proxies

Similar to Jini, .NET Remoting uses proxies to perform a variety of remote task depending on the architecture. In the Microsoft .NET Framework there are two classes that work together to form the Remoting proxy: `TransparentProxy` and `RealProxy` [16]. The client of a remote object instance always communicates with that remote object through an instance of a `TransparentProxy`, which gives the caller the appearance of the target object. The `RealProxy` class is the source of information about and communication with the actual remote object. A caller obtains an instance of the `TransparentProxy` class from the `RealProxy` in which the `TransparentProxy` is contained.

2.2.4. Leasing

A lease is an object that encapsulates `TimeSpan` value that the Remoting infrastructure user to manage the lifetime of a remote object. The Remoting infrastructure provides the `Ilease` interface that defines the functionality. When the runtime activates an instance of either a well-known `Singleton` or a client-activated remote object, it asks the object for a lease. When the client calls a method on a remote object, the .NET Remoting infrastructure will decide how much time remains until the lease expires.

3. Smart Proxy Architecture

In order to distribute service requests as evenly as possible, a Jini and .NET system must provide the following two features:

- Services can update their load information.
- Clients can obtain the load information of services and switch to less-loaded services easily.

This section presents a smart proxy architecture that provides these two features through the leasing models and proxies of Jini and .NET. Services will periodically update their load information to the Jini Lookup service when they renew leases, and clients will consult with the Lookup service to determine if it is necessary to switch to less-loaded Jini or .NET services. This approach can perform dynamic load-balancing of Jini and .NET services while not incuring extra network communications. Figure 5 depicts the system with dynamic load-balancing mechanism via smart proxies.

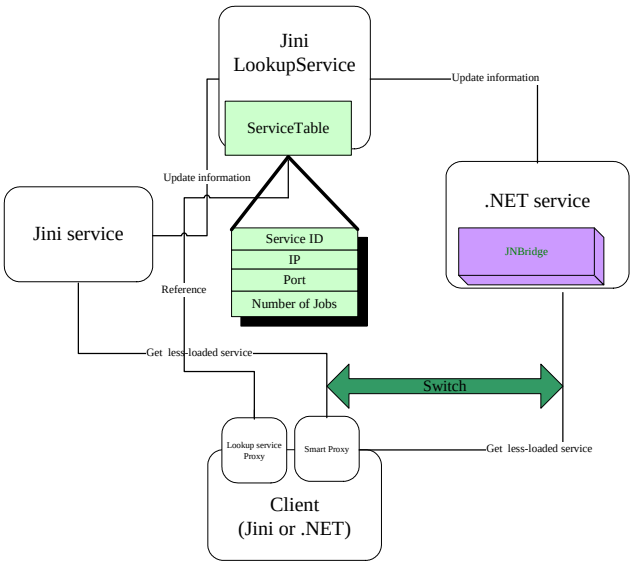


Figure 5. Smart Proxy Architecture

3.1. Integrating Jini and .NET via JNBridge

JNBridge is a Java/.NET interoperability tool that enables Java code to fully participate in the cross-language development capabilities of Microsoft .NET. Consequently, Java code can be called from .NET code and Java classes can be extended by .NET classes, and vice versa. However, .NET does not implement the main feature of Jini: a trio of protocols called *discovery*, *join*, and *lookup*. Specifically, .NET does not contain the Lookup services provided by Jini. Therefore, .NET services will now have to register themselves and upload their proxies to the Jini Lookup services.

3.2. Updating Load Information via Leases

A ServiceTable residing on the Lookup service keeps the load information of all services. Services must periodically report their load information to the Lookup services. However, each update action will initiate one extra network communication, even though the load information contains only few bytes, Such an overhead can be easily avoided since there is already a periodic communication existed between any service and the Lookup service— lease renewal. Therefore, the load information can be piggybacked onto the lease renewal object of services. As a result, no extra network communications will be incurred.

3.3. Switching Services via Smart Proxies

When a client first looks for a service, it will download a smart proxy from the Lookup service. Afterward, the client will request a service directly through the smart proxy, which will consult the load information on the ServiceTable and choose an appropriate service based on its load-balancing strategy (Section 4 will compare the efficiencies of several load-balancing strategies). That is, the client can switch freely between Jini and .NET services as shown in Figure 5 and the process is transparent to the client.

Host	CPU	Memory	O.S.	Clients	Services
S1	P4	512M	Windows 2k	2 Jini + 1 .NET	1 (Jini)
S2	P3	256M	Windows XP	2 Jini + 1 .NET	1 (Jini)
S3	P4	512M	Windows 2k	2 Jini	1 (.NET)

Table 1. Clients and Services

4. Experimental Results

The experiment platform consists of 3 network-connected computers, as listed in Table 1. Each computer hosts a Jini or .NET service that accepts requests from a pool of 8 Jini and .NET clients. In addition, a Lookup service is hosted by the first server, S1. The configuration of the experiment platform is depicted in Figure 6.

Experiments have been conducted for the configurations with different load-balancing strategies implemented in smart proxies, as listed in Table 2.

SP0 is the original prototype implementation of Jini and .NET, which does not have dynamic load-balancing mechanisms [15, 17]. Once a client has chosen a Jini or .NET

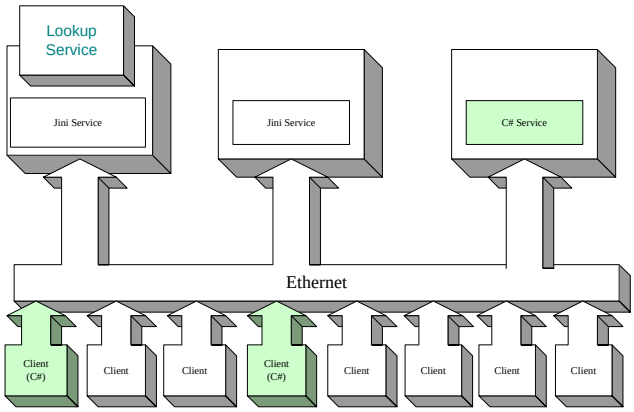


Figure 6. System Configuration

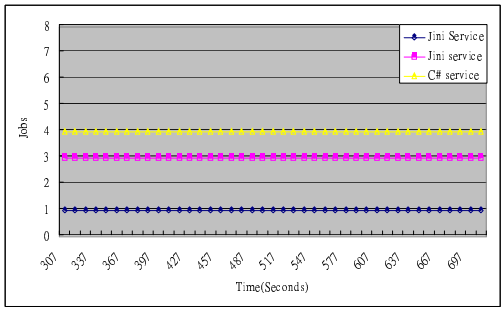


Figure 7. Numbers of Clients Served by Services

service, it will not switch to other services at all. Figure 7 shows that S1 has only 1 job all the time, while S2 and S3 are requested by 3 and 4 clients, respectively. This figure demonstrate that such a strategy leads to very unbalanced loads — the queue length of S3 is 4 times of the length of S1.

Any smart proxy in the SP1 configuration will always check the load information of all services and then make its client switch to the least-loaded Jini or .NET service. However, Figure 8(a) shows that the Jini and .NET system under SP1 is not balanced either. The reason is because the load information of services might not be up-to-date since services only update the information when leases are due. As a result, almost all the clients might switch to the same service that appears to be the least-loaded. Figure 8(a) reveals that frequently all clients request the same service, while leaving other services unoccupied. SP1_UB is a variation of SP1 that services will update their load information once an upper bound is reached, which is set to 3 in this paper. Consequently, load information of services will always be up-to-date and hence the loads are more balanced, as shown in Figure 8(b).

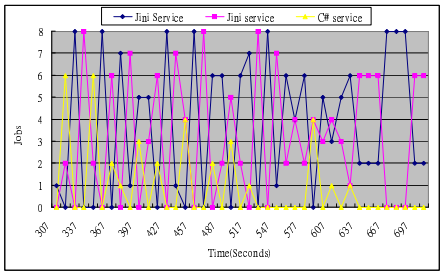
SP2 tries to avoid the situation of SP1 that all clients rush

Configuration	Load-Balancing Strategy
SP0	Original configuration of Jini
SP1	A client always uses the least-loaded service
SP1_UP	A client always uses the least-loaded service; A service updates its load information when the upper bound is reached
SP2	A client randomly pick one of the two least-loaded services
SP2_UP	A client randomly pick one of the two least-loaded services; A service updates its load information when the upper bound is reached
SP3	A client finds a service using a weighted voting strategy (weight = 1/queue length)
SP3_UP	A client finds a service using a weighted voting strategy; A service updates its load information when the upper bound is reached
SP3_UP_SC	A client finds a service using a weighted voting strategy (weights are scaled); A service updates its load information when the upper bound is reached
SP4	A client finds a service using a weighted voting strategy (weight = service rank)
SP4_UP	A client finds a service using a weighted voting strategy; A service updates its load information when the upper bound is reached
SP4_UP_SC	A client finds a service using a weighted voting strategy (weights are scaled); A service updates its load information when the upper bound is reached

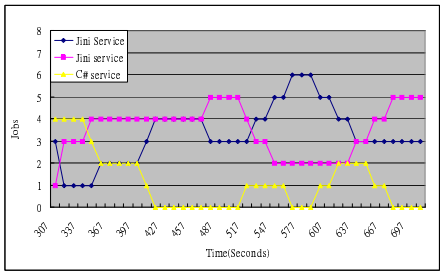
Table 2. Setup of Experiments

to one same service by choosing two least-loaded services and then randomly picking one for request. Figure 8(c) demonstrates that the Jini and .NET system under SP2 is better balanced than SP1. However, the services in the Jini and .NET system under SP2 might still have to serve up to 6 or 7 clients from time to time. Similar to SP1_UB, services in SP2_UB update their load information once a pre-set upper bound is reached, and the loads are better distributed than SP2, as depicted in Figure 8(d).

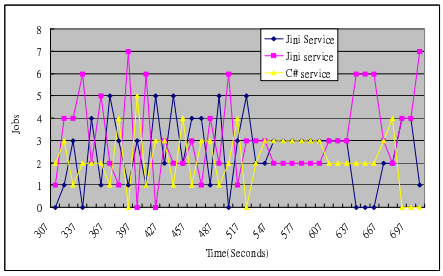
SP3 uses a totally different strategy from SP1 and SP2. Each service S_i uses the reciprocal of its queue length as its load, i.e. $L_i = 1/|S_i|$. The only exception is when the queue of any service is empty and the queue length will be set artificially to 0.1. When a client makes a request, its smart proxy will use a random number generator to pick a



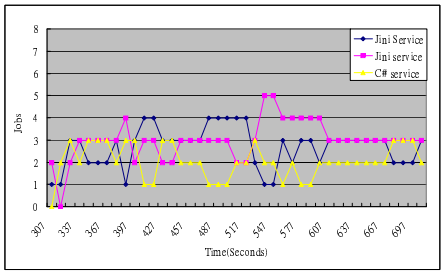
(a) SP1



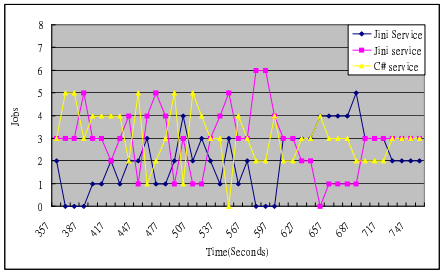
(b) SP1_UB



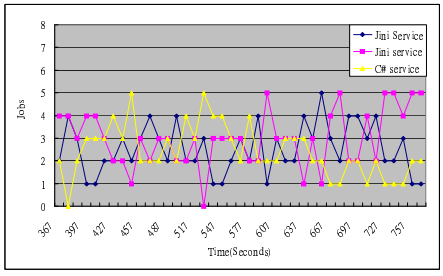
(c) SP2



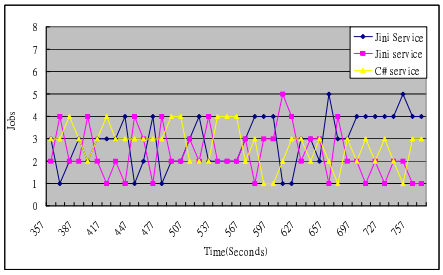
(d) SP2_UB



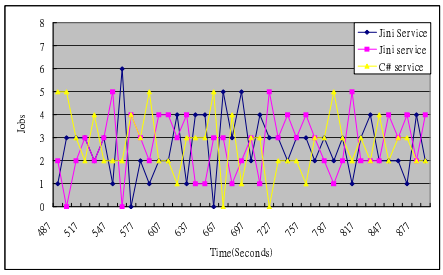
(e) SP3



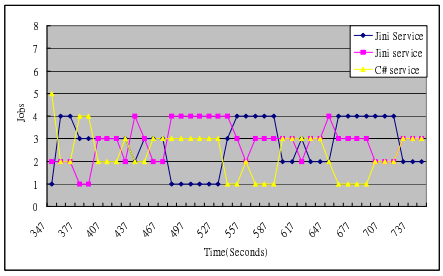
(f) SP3_UB



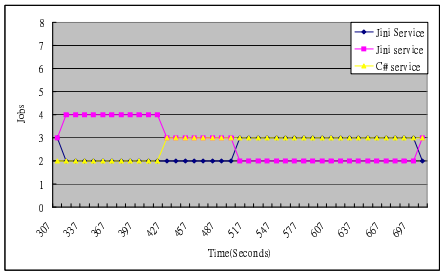
(g) SP3_UB_SC



(h) SP4



(i) SP4_UB



(j) SP4_UB_SC

Figure 8. Experimental Results

service and the service S_i will be chosen with the probability of $L_i / \sum_{j=1}^{3} L_j$. Figure 8(e) shows that SP3 performs better than SP1 and SP2. Similar to SP2_UB, services in SP3_UB update their load information once an upper bound is reached. In addition, SP3_UB_SC is a scaled version of SP3_UB, that is, the weight of the service S_i is computed as $L_i = 1/(2 \times |S_i|)$. Figure 8(f) and Figure 8(g) show the loads are very well balanced under SP3_UB and SP3_UB_SC.

SP4 is similar to SP3 and the only difference is how the load is computed. The load of service S_i is its rank in queue length, i.e. L_i =(service rank of S_i). In other words, the service with shortest queue length (ranked 3) will have a load of 3, while the busiest service will get a load of 1. Similarly, services in SP4_UB update their load information once an upper bound is reached, and SP4_UB_SC uses the square of the rank of service S_i as the scaled weight of S_i (i.e. L_i =(service rank of S_i)^2). Figures 8(h)∼(j) show that SP4 delivers even better load-balancing efficiency than SP3.

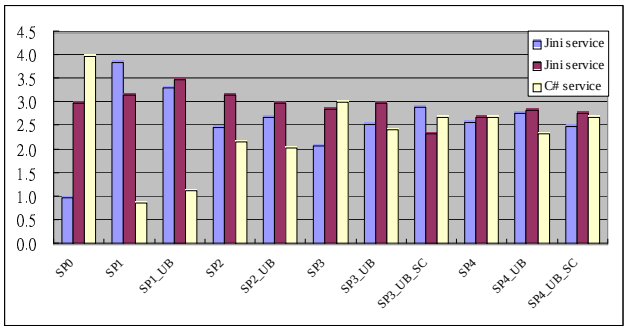


Figure 9. Average Numbers of Jobs

Figure 9 presents the average numbers of clients served by services under all configurations. It shows that the average numbers of requests from clients to S3 and S2 in the original Jini and .NET system are 4 and 3 times of the number to S1, respectively. After deploying the load-balancing strategy SP1, the situation gets worse since all clients rush to the same service that appears to be the least-loaded. On the other hand, it depicts that loads get better balanced when SP2 is adopted. In addition, the average queue lengths of services under SP3 in Figure 9 show services are well balanced. Finally, Figure 9 demonstrates that the strategy used by SP4 can lead to well-balanced distributions of Jini and .NET client requests.

Figure 10 shows the distributions of the numbers of jobs on services. It reveals that under SP0 and SP1, the numbers of jobs on servers are not balanced. On the other hand, it illustrates that under SP2, SP3, and SP4 dynamic load-balancing strategies services have spent most of time serving 2 to 3 requests from Jini and .NET clients, which indicates the Jini and .NET system is well balanced.

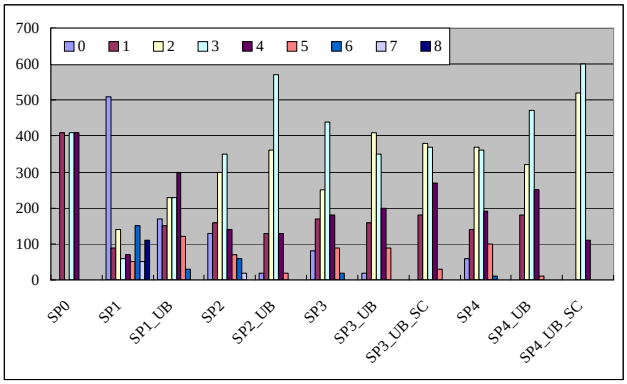


Figure 10. Distributions of Numbers of Jobs

5. Conclusions

This paper has integrated Jini and .NET into one distributed computing environment so that Jini and .NET clients can make request to Jini and .NET services. In addition, this paper has presented a dynamic load-balancing technique for Jini and .NET services. This technique uses a smart proxy architecture to dynamically redistribute Jini and .NET clients among Jini and .NET services based on its load-balancing strategy. In addition, the dynamic load-balancing process is transparent to clients. Several load-balancing strategies have been developed and the experimental results have demonstrated these strategies could distribute loads quite evenly. Furthermore, the load information of any service is updated when the service renews its lease with the Lookup Service. Consequently, updating load information does not incur additional network communications.

References

[1] K. Arnold. The Jini architecture: dynamic services in a flexible network. In *Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 157–162, 1999.

[2] M. Baker and G. Smith. Jini meets the grid. In *Proceedings of the 2001 International Conference on Parallel Processing Workshops*, pages 193–198, Sept. 2001.

[3] M. Beveridge and P. Koopman. Jini meets embedded control networking: A case study in portability failure. In *Proceedings of the The Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*, pages 11–18, Jan. 2002.

[4] L.-S. Cheung and Y.-K. Kwok. The design and performance of an intelligent jini load balancing service. In *Proceedings of 2001 International Conference on Parallel Processing Workshops*, pages 361–366, sep 2001.

[5] L.-S. Cheung and Y.-K. Kwok. A new fuzzy-decision based load balancing system for distributed object computing. In *Proceedings of 7th International Euro-Par Conference*, pages 183–190, 2001.
[6] W. K. Edwards. *Core Jini*. Prentice-Hall, 2nd edition, 2001.
[7] V. Georgiev and V. Getov. Assignment schemes for replicated services in jini. In *Proceedings. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 129–136. IEEE, 2002.
[8] S. Guest. *Microsoft .NET and J2EE Interoperatibility Toolkit*. Microsoft Press, 2003.
[9] R. Gupta, S. Talwar, and D. P. Agrawal. Jini home networking: A step toward pervasive computing. *IEEE Computer*, 35(8):34–40, Aug. 2002.
[10] P. Ledru. Smart proxies for Jini services. *ACM SIGPLAN Notices*, 37(4):57–61, April 2002.
[11] S. Li. *Professional Jini*. Wrox, 2000.
[12] H.-H. Lin. Load-balancing Jini services with smart proxies. Master's thesis, National Taiwan Ocean University, June 2004.
[13] H.-H. Lin, C.-H. Tu, and Y.-S. Hwang. Dynamic load-balancing of Jini services with smart proxies. In *Proceedings of The 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05), Vol. II*, pages 721–726, 2005.
[14] S. McLean, J. Naftel, and K. Williams. *Microsoft .NET Remoting*. Microsoft Press, 2002.
[15] Microsoft Corp. .NET: Driving business value with the microsoft platform. http://www.microsoft.com/net/default.mspx.
[16] J. Sievert. Create a custom marshaling implementation using .net remoting and com interop. *MSDN Magazine*, 18(9), sep 2003.
[17] Sun Microsystems, Inc. Jini network technology. http://www.sun.com/software/jini/index.xml.
[18] T. Urnes, A. S. Hatlen, P. S. Malm, and Ø. Myhre. Building distributed context-aware applications. *Personal and Ubiquitous Computing*, 5(1):38–41, Feb. 2001.
[19] J. Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, July 1999.