A Load Balancing technique through Jini in Distributed Processing

Rajeev Sharma [*], Rupak Sharma
Department of Computer ScienceEngineering, SRM University, NCR Campus, India

Article Info
Article history:
Received 4 January 2015
Received in revised form
10 January 2015
Accepted 20 January 2015
Available online 31 January 2015

Keywords
Remote Method Invocation,
Java Virtual Machine,
Uniform Resource Locater,
Time to Live Service,
Application Programming Interface

Abstract
Jini is a set of specifications that enables services to discover each other on a network and that provides a framework that allows those services to participate in certain types of operations. For an instance, take a Jini-enabled laptop into a Jini-enabled conference room and the laptop automatically be able to find and use the services of the conference room such as the laptop will automatically find the printer inside the room, seamlessly download any drivers required by the printer and will send its output to the printer. But Jini is not about hardware and devices. Jini is all about services.
Thus Jini not only allows hardware and applications to interact but also allows this interaction to happen in a dynamic, robust way. Jini software also gives network devices self-configuration and self-management capabilities. It lets devices communicate immediately on a network without human intervention.This paper will provide the facility to dynamic load balancing i.e automatically detection of hardware and load balancing .

1. Load Balancing

Distributing processing and communications activity evenly across a computer network so that no single device is overwhelmed. Load balancing is especially important for networks where it's difficult to predict the number of requests that will be issued to a server Busy websites typically employ two or more Web Servers in a load balancing scheme. If one server starts to get swamped, requests are forwarded to another server with more capacity. Load balancing can also refer to the communications channels themselves.

Load balancing is dividing the amount of work that a computer has to do between two or more computers so that more work gets done in the same amount of time and, in general, all users get served faster. Load balancing can be implemented with hardware, software, or a combination of both. Typically, load balancing is the main reason for computer server clustering

2. Introduction to Jini

Jini is a simple set of Java classes and services that has the potential to create its own revolution because it allows technology to be exploited in new ways. This is software for networking in all sorts of electronic devices, services, and applications, Jini lets them join up easily, seamlessly and gracefully - it is a sort of plug-and-play capability for spontaneously forming networks of heterogeneous equipment to share code and configurations transparently. And Jini has the potential to radically alter our use of computer service networks, since it allows and encourages new types of services and new uses of existing networks.

3. What is Jini

The Jini technology makes a network more dynamic by allowing the plug-and-play of devices. It provides mechanisms for devices to join and detach from network dynamically without the need for configuring each device.

The central mechanism of a Jini system is the Lookup service that registers devices and services available on the network. It is also the major point of contact between the system and the users of the system.

When a device is plugged into the network, it locates the lookup service (discovery) and registers (join) its service there. While registering, the device provides a callable interface to access its functionality and attributes those may be useful while querying the lookup from another 1 inni client perspective. Now, the service offered by the device may be located and used by clients on the network. This basically involves discovering lookup, querying it for the specific service and invoking the callable interface of the service required. The callable interfaces are exposed and accessed through Java RMI (Remote Method Invocation)[1]. The proxy code that is used to access an interface is stored within the lookup services during registration and downloaded by clients automatically. Thus the need for specialized drivers is eliminated.

Thus Jini allows building up clusters of services that know about one another and cooperate – creating a “federation” of devices. The term federation may be understood to be a collection of co-operating, but autonomous entities. However, with Jini, Java on the devices is almost mandatory, and the interaction mechanisms – the interfaces, the methods, what to call, when to call must be well defined for devices to join a federation that you create. In the next few sections, we try to explain Jini from the perspective of implementing a Calculator service that a Jini federation makes use of.

A Jini-enabled printer provides a print service to the network; that the particular service is provided by a piece of hardware is irrelevant. There may be a software service on the network that also provides the same print service by accepting print requests, rasterizing them and emailing the rasterized images to a distant recipient. There are many services that are only software.

Jini has the potential to radically alter our use of computer service networks, since it allows and encourages new types of services and new uses of existing networks. These networks are self-healing in that devices that leaves

Corresponding Author,
E-mail address: dcs.rajom@gmail.com

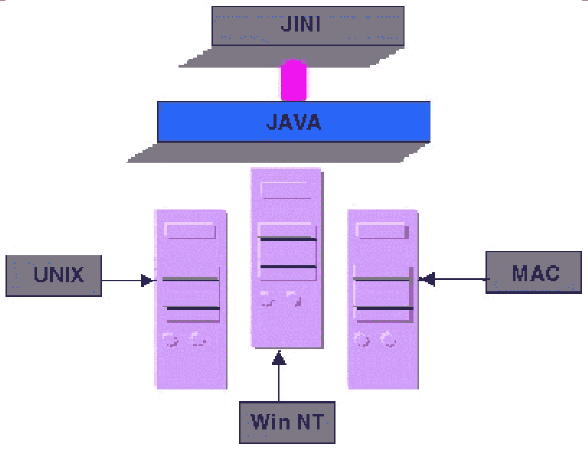

Fig: 1. Jini Architeture

the network for any reason, such as machine crashes or power surges, do not affect the remaining devices' operation. A Jini client that loses contact with a server can recover and continue processing. It is precisely these features that make Jini technology ideal for embedded systems in a dynamic environment. But network plug-and-play capabilities and self-configuration are also attractive for enterprise systems [9]

4. Advantages

Jini technology is a sophisticated platform on which to develop network-aware applications. Jini technology provides users access to resources located anywhere on the network. Both user and resource locations can change without affecting the application. Users, devices, and resources can join and leave the network without manual reconfiguration [3]. Jini developers used the Internet as a model for developing their product and sought to take advantage of the Internet's advantages in terms of reliability, scalability, maintenance and administration, and security. Jini is freed from having to deal with specific operating system and hardware requirements by Java technology, while Jini itself frees the client and service to interact without having to concern themselves with the particulars of the network [6].

5. Jini Infrastructure

Jini is a distributed computing framework. Hence the participants in the Jini network are called clients and servers. A server has an interface, which is the API that it presents to the outside world. This interface is called the service interface or the service. A server is hence an implementation of a service.

Jini software runs on top of Java Virtual machine and will work in any IP-based network of machines with Java VMs. It is based entirely on Java and depends on Java to function. Jini technology presupposes the existence of network connecting devices and Jini-enabled communicate with each other over this network. A Jini network contains communities, or federations, or clients and services. A Jini service joins a federation, to share its service with clients. A Jini client joins a federation to gain access to services. Federations are dynamic constructs, appearing and disappearing based on the demands of Jini devices.

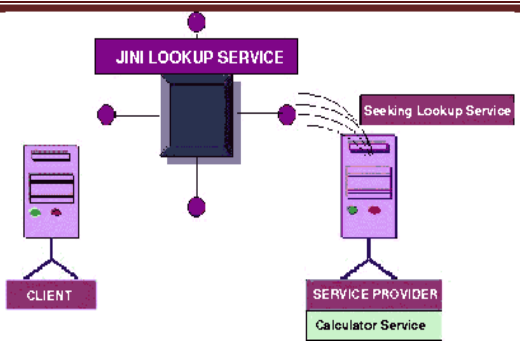

Fig: 2. Jini a Distributed Computing Framework.

6. Jini Lookup Service

The Jini Lookup service is the heart of a Jini community. The lookup service is similar in principle to the naming service used in other distributed computing paradigms like CORBA's COS Naming Service. It holds the registrations of all other services available in the Jini community. An application that wants to use a Jini service finds the desired service by looking for the service's registration within the lookup service [2]. A Jini service must register itself with the Jini lookup service in order to be used. The Jini lookup service is just another service. The service interface of the lookup service defines all the operations that are possible on the lookup service. It defines the way in which clients in a Jini community locate services. There may be more than one instance of the lookup service running in the community. This is to provide a certain level of redundancy. Jini lookup services are organized into groups. There is a default group called the public group. One can create groups with any name. When he starts a lookup service, he can specify which group it should hold registrations for. When he searches for lookup services, he can specify which groups he is interested in.

A. **Lookup Discovery:** A client that wants to use a particular service finds the appropriate server by looking in the Jini lookup service. But before that the client has to locate the Jini lookup service. This can be done by a process called discovery. Jini specification for this defines at the network level the protocol by which clients can find the lookup service. There are two ways in which a client can discover the Jini lookup service:

B. **Multicast discovery:** the client sends out a multicast request of a specified format. All Jini lookup services that come across the request will respond to it, and the client is said to have discovered the lookup services.

C. **Unicast Discovery:** In this, the client should attempt to connect to a lookup server having known the existence and location of the lookup server. This is in a sense not discovery. The discovery protocol uses a combination of both unicast and multicast discovery in order to find lookup servers.

Clients will discover all lookup servers that are within the multicast radius of the network. The multicast discovery packet will be broadcast on the local network. Routers that join two networks may or may not route the multicast packet between the two networks depending upon the rules of the router. Many routers will not pass any multicast packets between networks and for these networks; the multicast radius is equal to the local network. Otherwise, the router will forward the multicast packet based on its time to live (TTL) value, which is decremented each time the packet passes through a router[8]. When the TTL value reaches zero, the packet is no longer forwarded onto new networks.

7. Service-based Model

A service is a fundamental concept in Jini Technology and represents an entity that users can access over a network. Services can be a storage device, software component, printer, home stereo, or satellite navigation system. Basically, they are any item that programmers can abstract through software. Services advertise their capabilities through a lookup server, which Jini clients use to discover and access services. Jini devices that advertise services can themselves be clients of other Jini devices and through the dynamic combination of these services, Jini clients can build complete distributed applications [5]. For these reasons, the Lookup service is key to the Jini network. A Jini server specifies attributes, such as accounting System or license Server, to facilitate identification of particular services available in it.

8. How Jini Works

The first action of a Jini-enabled device after connecting to the network is locating the Lookup service, which resides on a server in a network. It accomplishes this by using the discovery protocol, which is multicast-based. For this, a device first sends a message to every other Jini device on the network that is within a certain number of network hops. When the Lookup server receives this broadcast request, its returns its address to the Jini client, which stores the address as its link to the Lookup server. The client thereafter sends requests to the Lookup server directly to this address. Similarly Jini servers offering services uses the discovery protocol, which is multicast-based, to find a Lookup Service and then uses the join protocol to register and thus become available to client applications. A Jini client asks the Lookup service for a list of Jini servers that match the requested attributes. It is then up to the Jini client to select the specific Jini server from the returned list [11]. Similarly Jini servers offering services uses the discovery protocol, which is multicast-based, to find a Lookup Service and uses the join protocol to register and thus become available to client applications. Services may join multiple lookup servers for increased reliability. Once clients and services are paired up, they no longer need the Lookup Service's assistance and can work together directly. The primary job of a Lookup service is to act as a central market for Jini services scattered around the network, grouping similar ones together and making them accessible to client application programs. The Lookup service maintains the maps between each Jini service and its attributes.

9. Server-side Processing

A unique feature of Jini technology is its ability to transfer executable code between Jini devices. This capability is similar to Java applets, which execute on Web browsers. With applets, Web servers download an entire applet executable code to the client, which executes the code. In contrast, Jini software sends only the code for the interface that the client uses to communicate with the server; the rest of the program remains in the server and actually executes there. Thus, in a way, Jini server teaches the Jini client how to communicate with it. This strategy mitigates one of the biggest problems with Java applets: their slow speed. The system downloads less because a Jini network usually only transfers the results of whatever code the server executes.

10. Leasing of Service

When the Jini client receives the lookup service reply that responds to its request for a service, it initiates a connection with the server. The Jini specification does not stipulate how an application should implement a service. The only requirement is that a Jini service must implement a specific interface, which the server supplies to the client. The client has no prior knowledge of any server and only knows how to communicate with the server through the given interface.

When a client accesses a resource through a service, it leases that service. A lease is a guarantee that the client may access that resource for a specific length of time. The client has the option of renewing the lease before its expiry. If suppose the service does not receive an extension request within the allotted period of time, the resources may be allocated for other clients. This prevents a network or client failure from keeping an indefinite lock on the resource. A nonexclusive lease permits other clients also to access the service at the same time. A lease could be exclusive, allowing only one client to access a resource at any time.

11. Transactions in Jini

A distributed environment must provide guarantees because of the vagaries of networks and servers. In this environment. Jini provides a transaction API for clients and services to use. Jini infrastructure helps clients and services adapting to a dynamic environment by deleting or creating references to each other automatically. It however does not address the problem of what to do when an error occurs. This is where a transaction comes in. For instance, a simple transaction entails making a debit in the customer's account while making a credit in the seller's account. This transaction fails if the customer's account has insufficient funds. In a distributed environment, such functions can take place often on different servers. Thus transactions ensure consistency across the network. Jini provides an interface to a transaction manager that knows how to keep track of all the operations and how to manage the parties in the transaction. This allows a series of operations to be grouped into a transaction so that the transaction is either committed or aborted. In a Jini transaction, there are three parties:

A. **Transaction Client:** It initiates and terminates the transaction. It does this by obtaining a transaction object, providing that object to a series of method calls that the client wants treated as an atomic unit, and then committing the transaction. Only the client can commit the transaction; the client, as well as any other party in the transaction, can abort the transaction. Jini provides a set of APIs that we will use to be a transaction client.

B. **Transaction Manager:** It oversees the entire transaction. The transaction manager provides the transaction object to the client and keeps track of all participants. When the client commits the transaction, the transaction manager ensures that all participants update their data correctly. The transaction object is leased by the transaction manager. The client must renew the lease on the object. Failure to renew the lease will cause the transaction manger to abort the transaction

C. **Transaction Participants:** The services to which the client provides the transaction object become transaction participants. Transaction participants have strict requirements with respect to how they must manage data within the transaction. While the client initiates the transaction, it is the transaction participants that allow the use of transactions, because the participants specify in their service interface that the client must provide a transaction object.

The Jini transaction system uses a two-phase commit protocol in conjunction with all the participants. Jini allows nestable transactions. A nested transaction or subtransaction is a transaction that has another transaction as its parent.

Writing a service that participates in a transaction is difficult. While the transaction participant interface is straightforward, the guarantees that are mandated by the transaction framework are difficult to satisfy. Fortunately, few services need to support transactions.

12. Java Spaces Service

Java Spaces is a Jini service that builds and maintains a database of Java objects. Java Spaces allows the storage and retrieval of objects via a standard lookup. In this way, Java Spaces facilitates group communication through Java object sharing. The database search is similar to a file name search. But the file name search is not descriptive enough and hence in Java Spaces, one can perform a lookup via a name or any set of attributes, which can vary for each object. The Java Spaces application programming interface contains three operations:

A. Write performs the create and update functions
B. Read lets programs get a copy of an object and
C. Take combines read and delete.

Apart from these, Java Spaces also supports notify, which lets a program know when an object that matches some criteria has been written into the space. Towards synchronization, Java Spaces can take an object. Once taken, the object is not available to any other program. To avoid human intervention for releasing a taken object from the clutches of program, which takes the said object and dies, Java Spaces add a transaction to the take operation. If the transaction is rolled back, that is , does not succeed, the object automatically reappears in the space. But while the transaction is open, the object remains unavailable to the take operation.

The Java Spaces programming model typically follows the same pattern of other database programming model. Applications or classes use the write operation to place object into the Java Space; they also remove objects from the space for processing using the take operation. When processing completes, using a write operation again returns the resulting object to the space.

Two possible uses for Java Spaces are as a sharable whiteboard or as a compute server. In the case of the sharable whiteboard, applications or classes write sets, which other programs can share, into the space. As a computer server, the space works as a generic server that allows distributed calculations. Conceptually, the Java Space is the repository for work. A master process writes objects into the Java Space. Slave processes, running on other systems take objects from the Java Space. They perform whatever calculation is needed and write the result back to the space. This system lets several machines solve a single problem, which is broken into smaller chunks, in parallel. This technique is inherently self-load balancing.

13. Security in Jini

Jini services currently use the same security model as all Java 2 programs. The Java 2 security model operates only when a security manager is installed into an application. By default, applications do not have a security manager. However, in order for RMI clients to be able to download code from RMI servers, a security manager must be installed. Thus all the Jini services have such a security manager. Also all code is given a set of permissions. This set is determined by a combination of the URL from which the code was downloaded and the entities that signed the jar file containing the class. The set of permissions associated with a particular class is loaded from policy file. Jini policy files are available for all its services.

14. Conclusion

Thus Jini's promise is not limited to the domain of network devices. It can be expanded to scanners, printers, phones, radios etc. Jini will provide more dynamic services to the client for network without configuring the hardware devices. This will handle the references dynamically in distributed environment. i.e. How client is using services in dynamic environment by remote references . This technique will solve the problem of doing self load-balancing. This will solve the problem of doing manual reconfiguration.

References

[1] Author:"Downing" intitle:"Java RMI: Remote Method Invocation", 1998 - IDG Books Worldwide
[2] Java RMI by William Grosso, Publisher: O'Reilly, Pub Date: 2001
[3] R. Steflik, P. Sridharan, Java IDL From Advanced JAVA Networking
[4] R. Steflik, P. Sridharan, Java RMI From Advanced JAVA Networking
[5] Art Taylor, J2EE™ and Beyond: Design, Develop, and Deploy World-Class Java™ Software
[6] F. Bachman. Technical concepts of component-based software engineering, 2000
[7] R. Balter, L. Bellissard, F. Boyer, M. Riveill, J. Vion-Dury. Architecturing and configuring distributed applications with olan, 1998
[8] P. Bengtsson, N. Lassing, J. Bosch, H. V. Vliet, Analyzing software architectures for modifiability, 2000
[9] J. E. Cook, J. A. Dage. Highly reliable upgrading of com-ponents. In International Conference on Software Engineering, 1999, 203–212
[10] P. Costanza. Transmigration of object identity: The programming language gilgul. http://citeseer.nj.nec.com/483031.html.

[11] C. E. Cuesta, P. de la Fuente, M. Barrio-Solorzano, Dynamic coordination architectur through the use of reflection.SAC 2001, Las Vegas, NV, ACM 1-58113-287-5/01/02:134–140, 2001