Toward Middleware-based Online Application Migration

Fang Deng, Qianxiang Wang, Jin Shao
School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
Beijing, China
Email: dengfang07, shaojin07@sei.pku.edu.cn, wqx@pku.edu.cn

Abstract—As an effective approach to maintain software system without interrupting the service, online migration has been applied for many goals, e.g. achieving a higher system performance, tolerating faults, and upgrading systems. In this paper, we propose a middleware-based approach for online application migration. Independent from the operating system, middleware not only provides information required to perform migration, but also offers a flexible and efficient way for online application migration. We implement our approach on the J2EE platform, one of the most important middleware platforms in recent years. A J2EE application migration framework (JAMF) is developed to demonstrate the proposed approach. The experiments with two typical examples show that our approach is feasible and effective.

Keywords- application; migration; application migration; online migration; middleware; component;

I. INTRODUCTION

Nowadays, many network-based software systems are required to be large-scale, high-performance, and non-interruption. Online migration is an important approach to ensure the quality of such software systems, e.g. achieving a higher system performance, tolerating known faults, upgrading systems, etc. Online migration approaches which focus on different migration entities have been explored for a long time.

Researches in [1-6] have studied the migration of component. Compared with component, application is a larger software entity. Online application migration enables the redistribution of applications among physical nodes without interrupting the normal servicing process. Online application migration is useful in many situations. For example, (1) to survive impending shutdown, e.g. some node may need to reboot to solve some problem; (2) to resolve network bandwidth problem, e.g. an application can be moved to another node which has wider bandwidth so as to reduce the response time for end users [1]; (3) to adjust the loads of different computing nodes, e.g. a node may need to transfer some of its applications to another node which has more computing resources; (4) when a new node is added, some applications can be moved to the new node; and so on.

Online application migration can be carried out through online operating system migration, which is usually based on virtual machine [7], [8]. However, since this approach moves the whole memory image, it results in a long migration time, and is suitable for local area network only.

In this paper, we propose a middleware-based approach for online application migration. This approach focuses mainly on the component-based applications which are usually composed of multiple kinds of units. For example, J2EE applications are usually composed of EJB, JSP, and Servlet.

The proposed approach has the following benefits:
• The migration process needn’t transfer the low-level state information associated with operating system and hardware, which is the way that virtual machine based migration does. Moreover, the data volume that needs to be transferred during migration is relatively small. Thus, it is possible to perform application migration between nodes in different networks, not only within local area network.
• Middleware-based approach makes the migration process easy to implement, even the low-layer platform is heterogeneous, because all of the information about the application to be moved can be obtained from the middleware easily.
• Middleware-based migration provides a way to move single application, while virtual machine based migration moves all the applications as a whole.

The remainder of this paper is organized as follows: Section II is an overview of our approach. In section III, we illustrate our approach mainly by introducing the J2EE application migration framework (JAMF). In section IV, the detailed migration mechanism is demonstrated, which supports online J2EE application migration without interrupting the service. Section V shows the implementation and experimental results. Section VI is related work discussion and section VII concludes this paper.

II. APPROACH OVERVIEW

To redistribute applications among different computing nodes, three main issues should be solved:
• What are the trigger conditions of application migration? As migration is usually performed to achieve higher system performance, tolerate faults, upgrade systems and so on, the first step for application migration is to collect performance and error related runtime information which will be used as the basis of migration decisions.
• What are the detailed migration policies? A migration policy acts as the guideline for application migration by answering migration related questions.
For example, according to the current runtime information, which application needs to be moved? How to choose the destination computing node?

- How can we move an application from one node to another while not interrupting the service and keeping the migration process transparent to the end users? The mechanism should be well designed to satisfy these requirements.

Generally speaking, the redistribution process of applications consists of three phases: (1) collecting runtime information; (2) making migration decision according to a pre-defined migration policy; (3) moving the application from the source node to the destination node.

A. Collecting the runtime information

In our approach, two main kinds of runtime information are collected to assist application migration. One is the low-level runtime information, e.g. CPU and memory utilization of a specific node, which are usually collected in traditional load balancing. The other is the high-level runtime information monitored by the middleware. Middleware-based monitoring can provide runtime information about specific applications [9]. The high-level information includes: the availability of services, correct rate, average throughput, average response time, instance number, etc.

B. Choosing migration policy

The migration policy concerns mainly the following aspects: when to start migration, how to choose the source node and destination node, and how to maintain the stability of migration process, etc. The administrator should configure the policy, for example, define the threshold for overload, choose the load balancing policy or just move whatever applications that one demands from one node to another.

The migration policy plays an important role in the migration process and should be designed elaborately to optimize the migration. For example, the required extra resources of the candidate for destination should be defined in the policy properly, so that the destination node will not get overloaded easily after the migration. Otherwise it will lead to another migration again in a short time.

C. Moving the application online

To perform online application migration, the main work is to capture all the data of the application on the source node and resume it on the destination node.

The data of an application are the set of every integral part of the application entity. The data of an application entity can be divided into static data and dynamic data. Static data include the byte code and configuration information. This kind of data, which are fixed once the deployment is finished and keep stable at runtime, are easy to obtain and recover. Dynamic data are produced in the running process, changing at runtime and complex to get and recover. Such data include: states of all kinds of instances, connections to the database, etc.

For the application that is composed of components, the core process of obtaining the data of the application is a process of traversing all the instances of every component. In this process, we need to ensure the consistency of the instance states before and after migration. For example, the state of a component instance should not be transferred when any of its methods is in an execution or a transaction, because the instance state changes frequently in method executions and the transfer of the state makes the rolling back action of a transaction hard to perform.

To achieve a reliable and efficient migration process, two main problems must be solved: (1) how to ensure that the application state keeps consistent and the application runs correctly during the migration process; (2) how to make the process transparent to end users and minimize the side effects on users.

III. DESIGN OF JAMF

In this paper, we designed a J2EE application migration framework (JAMF) to redistribute applications among different computing nodes online. JAMF can be implemented as a part of a J2EE-compliant application server. It collects the runtime monitoring information and provides an overall control for the migration behaviors among nodes according to a pre-defined migration policy. JAMF is made up of three parts, monitor, coordinator, and migration engine. Fig.1 is a general illustration of JAMF and the application redistribution process.

A. Monitor

The monitor runs on every node. It is used to:

- Monitor local runtime information.
- Report runtime information to the coordinator passively when the coordinator asks for it (pull-mode).
- Report to coordinator actively (push-mode) when pre-defined event happens or threshold is reached so as to launch a migration according to the migration policy.
- Receive messages of migration decision from the coordinator, pass the messages to migration engine, and contact the destination node to prepare for the migration work.

B. Coordinator

The coordinator runs only on the master node. It is used to:

- Collect runtime information from each node.
- Receive migration requests from the monitors.
- Make migration decisions according to the migration policy, and send messages to source node and destination node to start the migration.

To make a correct and efficient migration decision, the migration policy must be well configured. The policy should include information such as: in what cases an application on a particular node needs to be moved and in what cases a node is suitable to be selected as a candidate for destination node. For example, we could define that if one of the following thresholds is reached: the CPU utilization is over 70% or the instance number is over 5000, the node is considered not suitable to be a destination node.
C. Migration engine

The migration engine is the main part of JAMF, which runs on every node. Its function is to perform the actual behavior of moving an application from one node to another. The migration engine gets all the information needed about the specific application. Then migration engines on both source and destination nodes cooperate with each other to execute the migration and transfer the necessary data. In section IV, the detailed migration mechanism is described.

IV. J2EE APPLICATION MIGRATION MECHANISM

This section introduces the mechanism that implements the migration engine of JAMF. A J2EE application consists of EJB (Enterprise Java Bean) and web components. By this mechanism, a single J2EE application can be moved from one node to another without interrupting the service. The general process of moving an application consists of four phases:

- Initial transfer. Transfer byte code and configuration of the application which are usually in form of an EAR file.
- Application deployment. The application is deployed automatically on the destination node. Pre-deployment can be used to enhance migration efficiency.
- Dynamic data transfer. Capture the dynamic data of application from the source node and transfer them to destination node.
- Recovery. Recover the application state and resume the application on destination node. Meanwhile shut down the application on the source node.

Besides the migration process, two issues should be taken into account. First, how to maintain the consistency of the application state; Second, how to deal with the client requests, so as to make sure that they are not lost and get the prospective responses.

A. EJB migration

EJB is an important part in the composition of a J2EE application. In EJB 2.0 specification, there are four types of EJB: SFSB (stateful session bean), SLSB (stateless session bean), EB (entity bean) and MDB (message driven bean). The specific process of EJB migration is described as follow:

In the initial transfer phase, the byte code and descriptor of each EJB are transferred to the destination node.

In the application deployment phase, each EJB is deployed on the destination node and starts to provide service. But they are not available for requests from old EJB clients. Here “old EJB clients” refer to those clients that connected to the source node before migration and are relocated to the destination node now. When these beans become available to requests from the old EJB clients will be discussed in the second subsection.

In the dynamic data transfer phase, we visit every EJB container and deal with all the instances in each container. For each instance, we get its state and transfer it to the destination node. Different actions are taken to deal with instances of different EJB types. For EB, we submit the state into the database where the state of an EB is usually maintained. For SLSB and MDB, there is no state information. For SFSB, the state of the instance is passivated to byte data, and transferred to the destination node. The passivation action is carried out by invoking the ejbPassive() method defined in the javax.ejb.EntityBean and javax.ejb.SessionBean interfaces. Both entity bean and session bean are restricted to implement ejbPassive() and ejbActive() methods according to EJB 2.0 specification, and this ensures that it is not necessary to deal with external resource references such as database connections.

In the recovery phase, the states of EJB are recovered on the destination node. Specifically, EB reads the state from the database; SFSB recovers its state by accessing the transferred state data. The state recovery action is carried out by invoking the ejbActive() method. Last, we shut down the EJB containers on the source node. And the EJBs on the destination are ready to serve for all clients (both old and new EJB clients).

In the process of migration, there are some key problems to solve which will be discussed in the subsections

1) Preventing the loss of EJB client requests: When an application is moved to a new node, the requests from
connected EJB clients to the source node are likely to fail to get the responses. Our solution is to employ an automatic stub updating mechanism.

In EJB 2.0, clients access EJBs through home and remote objects. The actual object that a client gets is a stub of the home or remote object. Because the information which is needed to connect to the server is contained in the stub, the stub will be replaced with a new one when the client finds out that the application has been moved to another node. With the new stub, it can connect to the destination node successfully. The new stub can be obtained from the global naming service or we could make container on the source node to provide it. The whole process is automatic so that the client program will not feel.

2) Maintaining the consistency of instance state: The problem of inconsistency of instance state is likely to occur when a request arrives during the process of moving stateful session beans. This problem can be solved by state and request control at instance-level. When stateful session beans are moved, the source node does following things:

(1) Mark an instance with “transferred” tag.
(2) Passivate the instance state to byte data. Wait, if the bean is in a method execution or a transaction.
(3) Transfer the passivated state to the destination node. Go to (1).

To maintain the consistency of instance state, when a request arrives, different actions will be performed in different transfer periods of an instance state. Fig. 2 shows the three transfer periods and a client sends request A, B, C sequentially in these periods.

During the migration process, if a connected EJB client sends a request to the source node, the tag of the instance state will be checked first:

- If it is not marked with “transferred”, it indicates that the instance state has not been transferred yet. In this case, the client is still served by the source node as illustrated by request A and response A in Fig. 2.
- If it is marked with “transferred”, it indicates that the instance state is in its transfer process or has already been transferred to the destination node. Then, the stub is updated and the client tries to send the request again to the destination node automatically. This is the case when request B happens in Fig.2. The stub updating happens only once, so the latter request C does not lead to stub updating again.

Meanwhile, at the destination node, if a request is from an old EJB client that previously connected to the source node, the state of the corresponding instance will be checked. If it is not ready, the request will be blocked at the destination node. For example, in Fig.2, the redirected request B is blocked until the transfer process is finished while the request C can get its response immediately. It is worth noticing that control at instance-level shortens the blocking time effectively because a client is redirected to destination node after its state starts to be transferred and is released as soon as the transfer finishes. It does not need to wait for the whole migration process to finish.

B. Web component migration

The migration process of web component undergoes the four phases too, which is similar to that of EJB. The state passivation and recovery are done by the web container. For example, in Tomcat web server [10], methods to manipulate the web session are provided by a session manager and can be invoked through JMX (Java Management Extensions) interface. To prevent the loss of http requests, the web server is responsible to redirect the requests from source node to destination node. In Tomcat, the filter mechanism could take charge of the redirection. To maintain the consistency of the session state, similar approach as discussed in the second subsection of IV.A is employed to deal with every web session state.

V. IMPLEMENTATION AND EXPERIMENT

A. Implementation

JAMF is implemented as a part of PKUAS, which is a J2EE-compliant application server [11] and integrates Tomcat as its web container. JAMF can be managed by the management module of PKUAS. Thus, the migration policy can be configured through the unified user interface of the management modules. System information associated with the migration process can be obtained from the console.

B. Experiment

We have done two series of experiments on the JAMF. The first series is to calculate the average time expense of application migration. The second series is to evaluate the effects on client response time.

The two experiments are both conducted in a cluster of three hosts which are used as the master node, the source node and the destination node, respectively. Every host is equipped with an Intel Pentium IV 2.4GHz processor, a 512MB RAM, an Intel Pro 100/1000 Ethernet NIC in a 100M LAN, and a single 80GB 7200 RPM SATA disk. The database of MySql 5.1.6 is deployed on the master node. All the three nodes are deployed with PKUAS and applications.

![Figure 2. Requests from an EJB client which happen in different transfer periods of an instance state and their responses.](image-url)
1) Time cost of migration: We first measure the performance of JAMF in terms of time expense of migration. The test cases are two sample applications which are described in Tab.1. One is an online shopping store which allows people to purchase goods online. The first application consists of both web components and EJB components. The users access web pages through web browsers and the web components interact with EJB components. Another application is a chess game which is composed of only EJB components. The player runs a local Java program as the client which interacts with the application.

As illustrated in Fig.3, the total migration expense consists of two parts: (1) data transfer expense, which is made up of the time for the data transfer through TCP/IP, I/O work of state data and other initial work; (2) passivation expense, which refers to the time used for obtaining the state of the application. Pre-deployment is performed to enhance the performance and is not included in the total expense calculation.

Fig.4 shows that the total and passivation expense almost increase linearly when the client number of each application increases. On average, 69.38% of the total expense is taken up by passivation and 30.72% is taken up by data transfer.

2) Effects on client response time: We use the chess game application as our test case to evaluate the effects of migration on client response time. The effects are caused mainly by the stub updating and request blocking. We simulate 100 concurrent clients which send continuous requests to the application. The maximum, average, and minimum response time of each moment are recorded in the migration process as illustrated in Fig.5. When the migration period begins, the maximum response time increases because the instance states that some clients try to access are being transferred and these clients must update their stubs and be blocked for a while until the transfer process finishes. Since the instance-level control is used to optimize the migration, we can find obviously that the minimum response time remains normal and the average response time increases just a little (9.7 ms on average according to our statistic). Due to the instance-level control, although the migration begins, most of the clients are still served by the source node and do not need to update the stubs until their corresponding instance states are transferred. After the migration finishes, the stub updating will continue and the maximum response time returns to normal level after all the clients have updated the stub.

From Fig.5, we observe that the minimum response time is not affected by the migration and the average response time raises only a little and returns to normal level after all the EJB clients have updated the stubs. Although the maximum response time increases during the migration, each affected client suffers from this latency only once because the stub updating happens only once for each client. For user interactive application, the whole migration process is not noticeable to most end users.
VI. RELATED WORK

For component migration, Stéphane explores EJB migration which mainly concerns whether there are components associated with each other that need to be moved together and how to maintain the reference relationship between components and other resources [3]. Fan defines some kinds of components migration constraints which are used to maintain the component consistency during a component migration and gives a component migration algorithm [4]. But there are some drawbacks in the algorithm that still possibly results in request loss when it moves the cached requests.

Some researches focus on the migration of service. Meehean captures the service state through logging technique which is used in a distributed scheduling service [12]. But the implementation concentrates on the distributed scheduling service and cannot be reused for other services. Fu presents a service migration mechanism which is incorporated into a Java compliant distributed virtual machine, DSA [13]. They move and reconstruct the execution context of each process/thread of the service by operating the JVM runtime data structures including the heap, method area, and JVM stack. Sultan implements service migration with the support of modifications to the kernel of the operating systems [14]. The kind of service that they focus on is server application such as Apache Web Server and the PostgreSQL transactional database server. Both works in [13] and [14] need support of low-level mechanism such as heap, method area and stack of JVM or operating systems.

In [7] and [8], virtual machine technology is used to achieve operating system online migration. Operating system migration usually requires obtaining a large amount of complicated states through accessing hardware, or the virtual machine that the operating system resides on. It is limited by heterogeneous hardware and needs the support of uniform file access technology. The whole memory image has to be transferred, which consumes large amount of resources.

The approach we proposed does not rely on the operating system, lower virtual machine or hardware like those works mentioned above. It suits for migration across different networks with relatively narrow bandwidth while the operating system migration based on virtual machine normally adapts to local area network. Also our approach can move a few of the applications that run on the application server while all of the services provided by the operating system have to be moved as a whole at a time in operating system migration.

VII. CONCLUSION

In this paper, we proposed a middleware-based approach for online application migration. The middleware not only provides sufficient information required to perform migration, but also offers a flexible and efficient way for online application migration. With this approach, we can change the mapping relationship between applications and the physical nodes on demand automatically. The experimental results show that JAMF maintains the consistency of the application and can keep the application provide service continuously during the migration. The cost of application migration is conspicuously low and the whole process is transparent to most end users. Our ongoing work includes: integrating approaches with different migrating units to achieve a better solution, and optimizing the migration policy to enhance the autonomy of migration.

ACKNOWLEDGMENT

This paper is supported by National Natural Science Foundation of China, No. 60773160, the National Grand Fundamental Research Program of China No. 2005CB321805, and the National High-Tech Research and Development Plan of China No. 2006AA01Z175.

REFERENCES