Pattern-based J2EE Application Deployment with Cost Analysis

Nuyun ZHANG, Gang HUANG, Ling LAN, Hong MEI
Institute of Software, School of Electronics Engineering and Computer Science, Peking University,
Key laboratory of High Confidence Software Technologies (Peking University), Ministry of Education,
Beijing, 100871, China
zhangny04@sei.pku.edu.cn, huanggang@sei.pku.edu.cn, lanling@pku.edu.cn, meih@pku.edu.cn

Abstract

The most challenging problem of J2EE application deployment is to determine which components should be deployed onto which servers in terms of non-functional properties. This paper proposes an empirical approach with some mathematic enhancement. It does two contributions to J2EE application deployment: 1) patterns are adopted to specify why, when, where and how to deploy so that the empirical approach becomes more comprehensible and operational; 2) a mathematic framework to analyze the deployment cost is defined for helping the selection of the best-of-breed pattern. We experiment several patterns on a J2EE benchmark application and evaluate the cost analysis framework.

1. Introduction

Deployment of J2EE application is the process to deploy application clients, applets, web components and Enterprise JavaBeans components into specific operational environments [6].

The most challenging problem of deployment is component arrangement, namely which components should be deployed onto which servers in terms of performance, reliability, cost and other non-functional properties. Existing approaches for the problem can be summarized as: mathematics, artificial intelligence (AI) and empirical methods.

The mathematics methods build delicate models about the system and solving it to find the optimal deployment plan [4]. They require precise parameters, such as memory consumption of each component and reliability of each node. The AI methods use AI technology to reduce the complexity of the search space that is involved in finding an optimal plan [5]. They need a number of parameters, too. The empirical methods try to conclude some guidelines from experiences or best practices for deployments.

The math and AI methods neglect some details of the running of the application system. In addition, some of the parameters they need are difficult to measure or their precision cannot be guaranteed. Totally depending on them is usually impractical. On the other hand, when turning to the empirical methods, we find their carrying out is seriously depend on the deployers’ understanding and manual operations.

This paper proposes a pattern based approach for J2EE application deployment. It is an empirical method but more formal and specific than the traditional ones. It includes clear operational steps and can be carried out automatically. However, there are 2 problems in the approach: how to choose a pattern to execute from several useful patterns and how to map the deployment solutions onto physical machines. We again propose a deployment cost analysis framework to meet the problems.

The contribution of this paper is: it not only brings forward a pattern based approach to J2EE application deployment but also makes it practical. It makes use of patterns to describe deployment guideline and gives a quantitative cost analysis framework to support the execution of patterns. It implements the pattern based deployment process and validates the effectiveness of the approach and the framework by experiments.

2. Approach Overview

2.1 Description of Deployment Pattern

The description of deployment patterns contains several sections which are shown in Table 1.

2.2 Pattern based Deployment Process

We divide J2EE deployment process into seven stages that are release, installation, update, adaptation, activation, deactivation, and uninstallation. The pattern based J2EE deployment adds some steps between the stages as shown in Figure 1. The execution of pattern starts after the configuration step. Then it goes on as follows:

Firstly, the pattern execution tool identifies the pattern candidates by checking the goal sections and context sections of all patterns.

Secondly, the pattern execution tool picks out one of the pattern candidates to execute. It follows the solution section to make a deployment plan, the implementation section to make a physical deployment plan.
Table 1  Description of Deployment Pattern

| Section      | Purpose                        | Content                                                                 | Examples                                                                 |
|--------------|--------------------------------|Adam insertion test 1                                                                 | Adam insertion test 2                                                                 |
| Name         | Unique identification for the pattern | A meaningful name                                                                 | Components collocating pattern                                                                 |
| Goal         | For automatic recognition of the pattern | WHY: The goal of deployment                                                                 | short response time.                                                                 |
| Context      | For automatic checking of the pattern | WHEN: Preconditions for using the pattern                                                                 | Workloads, resource constraint of nodes.                                                                 |
| Solution     | For automates execution of the pattern | WHERE: Relationships between components and nodes                                                                 | Component A is on node n. Component B is on node m.                                                                 |
| Implementation | For automatic execution of the pattern | HOW: Relationships between components and physical machines                                                                 | Component A is on node n. Component B is on node m. Node n is Dell PC 1. Node m is Dell PC 2. |

Figure 1. Pattern based deployment process

Thirdly, the deployment tool packages the components and transfer them to their destinations according to the physical deployment plan.

Fourthly, the application servers install the packages and start them. If this is a redeployment, the application servers will stop and uninstall the old components before the installation and starting.

Finally, after a period of running, the tools check whether the deployment has the expected results and adjust the parameters in the patterns. The tools may start a new round of pattern based deployment process to improve the deployment.

3. Deployment Cost Analysis Framework

The challenges in pattern based deployment process are: 1) in the implementation section of a pattern, we should map the solution to the physical machines. 2) If useful patterns are more than one, we should select one for execution under circumstances that we do not know which pattern will make the system work most well.

There could be many kinds of solutions to the challenges. We consider in an enterprise application like J2EE application that the Cost/Benefit is most important. Since we have no idea of how many benefits deployment will bring, we try to make the cost of deployment minimum. In the deployment process, the service is stopped and that adds to the cost. Therefore, we proposed a framework to evaluate the cost of J2EE application deployment process.

We define the deployment cost as the time deployment takes. In the deployment or redeployment process, loses are brought by the system down time. So we care about it instead of the other expenses such as memory or network occupation.

Our goal is to find the pattern whose deployment time is the least. For this purpose, the cost analysis framework only care about the cost differences in deployment processes, not their absolute values. Some constants will be neglected to simplify the framework.

Packaging and transferring do not interrupt the running of the application system. We do not include them in deployment cost analysis.

The analysis framework is as follows:

The deployment cost on a node is composed of undeploy time of old packages and deployment time of the new ones, as shown in equation (2).
\[ \text{costOnEachNode} = \text{deploymentTime} + \text{undeploymentTime} \] \[ \text{(2)} \]

1. The deployment time is closely related with the implementation of application server. We have done some experiments on Peking University Application Server (PKUAS) \([3]\) try to find some rules in deployment time.

The deployment time is composed of installation time and starting time. The following hot color map is a contour figure showing the installation time of JAR packages. The 20 colors evenly ranged from 3396 to 17626 ms. The darkest red in the upper right area of the map represents the longest time that is 17626 ms. The darkest blue in the lower left represents the shortest time that is 3396 ms. There is a blank area in the right lower part of the figure, because when package number is small, the package size can hardly be very large.

![Figure 2. Contour map of installation time of JAR](image)

From figure 4 we can see the installation time has something to do with package size and number of JARs. The more JARs are, the longer the time is. The larger the size is, the longer the time is. Because the contour lines are almost vertical, we know the size of package plays a more important role in the installation time and their relationship can be approximately described by linear equation.

In our cost analysis problem, what we want is only a comparable result, not the absolute number. To simplify the comparison, we give each installation time a reasonable value instead of its real value. The given value can reflect its real value in proportion. The simplified expression of installation time of EJB package should be in a linear form, and we omit the constant to get the following expression:

\[ \text{installationTimeEJB} = \frac{100 \times \text{ejbpackageSize (kB)}}{100} \] \[ \text{(3)} \]

If necessary, by curve fitting and further analysis of the sub deployment time, we can get the exact relationship of installation time, the JAR number and the package size.

We can give a set of test packages. In another specific environment, by deploy the test packages, we can figure out the relationship of installation time, the JAR number and the package size automatically.

2. We do the same kind of experiments to figure out the installation time of WAR packages. In our experiments, the installation time of WAR differs from 282 to 33829 ms as the package size ranges from 41 to 64634 kB. The installation time of WAR is about ten times less than that of JAR of the same size.

We find that no matter how many html, JSP, and images are in the package, the installation time is the same if the whole package size is the same and the Java class number is the same. Figure 3 shows the installation time of WAR when there are a fix number of Java classes. We can see that the installation time has a linear relationship with the size of WAR package.

\[ \text{installationTimeWAR} = c \times \text{WARsize} \]

where \(c\) is a constant that has something to do with the implementation of the application server and the Java classes in the package.

![Figure 3. Installation time of WAR](image)

For the purpose of comparison, consulting the experiment results of the installation time of EJB package, we give a simplified expression for WAR installation time:

\[ \text{installationTimeWAR} = 0.1 \times \frac{\text{WARsize}}{\sum \text{JARsize}} \] \[ \text{(4)} \]

where \(\sum \text{JARsize}\) represents the size of all the JARs in the system.

Integrating the expression \((3)\) and \((4)\) we get the installation time of a package as:

\[ \text{installationTime} = \left[ \frac{\text{ejbpackageSize (kB)}}{100} + 0.1 \times \frac{\text{WARsize}}{\sum \text{JARsize}} \right] \] \[ \text{(5)} \]

3. By experiments we know the starting time is really small and can be neglect.

4. The undeployment time is composed of the uninstallation time and the stopping time. By experiments we know the stopping time is very small and can be neglected.

5. In our experiments, the uninstallation time of WAR varies from 188 to 781 ms as the package size ranges from 41 to 64634 kB. It plays a so trifling part in the whole deployment time that we neglect it.
Figure 4. Uninstallation time of JAR

6. Figure 4 is the contour map of uninstallation time of JAR packages by experiments. Because the contour line is almost horizontal, we know the number of JAR is the most important factor in uninstallation time. Using the same analysis method of the installation time, we get the following expression:

\[
\text{uninstallationTime}_\text{JAR} = 0.2 \times \text{JARnumber}
\]

In conclusion, applying the expression (5) and (6) to (2), we obtain the deployment cost on each node. Applying the expression (2) to (1), we obtain the cost of the whole deployment process.

In different implementations of J2EE deployment process, the cost analysis methods may need adjustment in detail, but the framework is still applicable and it is our future direction of research.

4. Case study

In this section, we use cost analysis based deployment patterns to redeploy RUBiS [7], which is an auction site prototype modeled after eBay.com that is used to evaluate application design patterns and application server performance scalability. First, we give an entire redeployment process of RUBiS. Then, we verify our cost analysis framework by comparing experimental deployment time and its estimated values. The process is aided by CADTool, which is a J2EE deployment tool [2].

The environment of experiments is shown in Table 2:

Table 2. Deployment environment

<table>
<thead>
<tr>
<th>Node</th>
<th>Software</th>
<th>Database</th>
<th>CPU</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aster0</td>
<td>Windows XP + PKUAS</td>
<td>none</td>
<td>2.79G</td>
<td>504M</td>
</tr>
<tr>
<td>Aster2</td>
<td>Windows XP + PKUAS</td>
<td>MySQL 4.0.15</td>
<td>2.99G</td>
<td>504M</td>
</tr>
<tr>
<td>Aster3</td>
<td>Windows XP + PKUAS</td>
<td>none</td>
<td>2.79G</td>
<td>512M</td>
</tr>
<tr>
<td>Client</td>
<td>Windows XP</td>
<td>none</td>
<td>2.79G</td>
<td>512M</td>
</tr>
</tbody>
</table>

4.1 Redeployment of RUBiS

4.1.1 Initial Deployment of RUBiS

In the given environment, there are 18 = 2592 deployment plan of RUBiS, where 18 is the component number and 3 is the node number. We make an initial deployment randomly, as shown in Table 3. We assume the number of concurrent clients is 500.

Table 3. An initial deployment plan

<table>
<thead>
<tr>
<th>Node</th>
<th>Deploys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aster0</td>
<td>WAR(241kB), 4 JARs(262kB)</td>
</tr>
<tr>
<td>Aster2</td>
<td>MySQL DB, 7 JARs(369kB)</td>
</tr>
<tr>
<td>Aster3</td>
<td>6 JARs(369kB)</td>
</tr>
</tbody>
</table>

4.1.2 Identification of Pattern Candidates

1. Deployment goal recognition. The deployment goal is to achieve a good performance in average client session time, as mentioned in section 2. We find out 5 patterns have the goal. They are in Table 4.

Table 4. Pattern Candidates

<table>
<thead>
<tr>
<th>ID</th>
<th>Key elements in context</th>
<th>Main idea of Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Client Number&lt;1000</td>
<td>Collocating WAR and JARs</td>
</tr>
<tr>
<td>2</td>
<td>Client Number&lt;1000</td>
<td>Centralized deployment of EJB</td>
</tr>
<tr>
<td>3</td>
<td>Client Number&gt;1000</td>
<td>Separate WAR and JARs</td>
</tr>
<tr>
<td>4</td>
<td>Client Number&gt;1000</td>
<td>Distributed deployment of EJB</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Separate Resource consuming servers</td>
</tr>
</tbody>
</table>

2. Initial context verification. In the experiment environment, the logic expressions in the contexts of Pattern 1 and 2 are true. Pattern 1 and 2 are useful.

4.1.3 Cost Analysis Aided Pattern Execution

1. Selecting an implementation for each pattern candidate.

We use the cost analysis framework to estimate the every possible implementation of the candidate pattern 1 and 2. We use the following formulae mentioned in section 3 to make the estimation:

\[
\text{CostOfApplication} = \max \{\text{costOnEachNode}\}
\]

\[
\text{installationTime} = \frac{\text{ejbpackageSize} \times 100}{100} + 0.1 \times \sum \frac{\text{WARsize}}{\text{JARsize}}
\]

\[
\text{undeployTime}_\text{JAR} = 0.2 \times \text{JARnumber}
\]
Table 5. Costs of the implementations of candidate

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Implementation</th>
<th>Cost on Aster0</th>
<th>Cost on Aster2</th>
<th>Cost on Aster3</th>
<th>Maximum Cost</th>
<th>Time by experiment (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Put all on Aster0</td>
<td>12+0.02, 0.2*4</td>
<td>0, 0.2*7</td>
<td>0, 0.2*6</td>
<td>12.82</td>
<td>14496</td>
</tr>
<tr>
<td>2</td>
<td>Put all on Aster2</td>
<td>0, 0.2*4</td>
<td>12+0.02, 0.2*7</td>
<td>0, 0.2*6</td>
<td>13.42</td>
<td>15695</td>
</tr>
<tr>
<td>3</td>
<td>Put all on Aster3</td>
<td>0, 0.2*4</td>
<td>0, 0.2*7</td>
<td>12+0.02, 0.2*6</td>
<td>13.22</td>
<td>15476</td>
</tr>
</tbody>
</table>

The estimation processes and results are in Table 5. Taking pattern 1 for example, it has 3 implementations. They are to put all the components on Aster0, Aster2 and Aster3. By cost analysis we know the deployment cost on Aster0 is the least, therefore we take it as the implementation of Pattern 1. By the same way we take deployment on Aster0 again as the implementation of Pattern 2. The last column in 5 is the experiment result for the 6 implementations. We can see our cost analysis framework do find the implementation with least deployment cost for each pattern.

5. Conclusions

This paper proposes a pattern based approach of J2EE application deployment and make it practical by a quantitative method to evaluate deployment cost. It gives experiments to validate them.

The idea of deployment pattern is not novel, but there is little attention paid in using of them, especially from the deployment cost point of view. To our best knowledge, there is no other work that does the same thing as we do, i.e., giving a deployment cost analysis framework of J2EE application by time.

The current weaknesses of the approach are: lacks of tool support and more analysis experiments on other kinds of application servers. The future work can be on studying more implementation of J2EE deployment process to find more common rules about the deployment cost. In our study of PKUAS deployment, an enhancement of the cost analysis precision is also needed.

References