Failure Analysis of Open Source J2EE Application Servers

Junguo Li, Gang Huang, Jian Zou and Hong Mei
School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China
{lijg05, huanggang, zoujian04}@sei.pku.edu.cn, meih@pku.edu.cn

ABSTRACT
Open source J2EE application servers (J2EE AS) have already attracted many business attentions in recent years, but reliability of open source products seems still an obstacle to their wide acceptance. The detail of software failures in open source J2EE ASs is seldom discussed in the past, despite such information is valuable to evaluate and improve the reliability. In this paper, we present a measurement-based failure classification and analysis of open source J2EE ASs, specifically taking J2EE services as basic units because a J2EE AS can be viewed as a set of common services. Presented results indicate there are still a lot of work to do to improve both reliability and user experience of open source J2EE ASs. Several major failure characteristics that may be of benefit to reliability improvement are: (1) a widely used fault-tolerant mechanism, clustering, is not reliable enough, and even likely to be a significant source of failures (account for about 18% of total failures); (2) only 15% of bug-caused failures can be removed within a week, and the worse is that about 10% failures are still open (without available patches); (3) exception and error catching would be a major error-detect approach that could monitor 64% potential failures in the best-case. Some other detect approaches would be helpful for the other failure manifestations; (4) more application specific efforts should be taken to deal with service semantic related failures which account for about 75% of all failures. Finally, we give a review of failure analysis approaches based on bug repository, and discuss their limits.

Categories and Subject Descriptors
D.2.5 [Software Engineering]: Testing and Debugging; D.2.4 [Software Engineering]: Software/Program Verification

General Terms
Reliability

Keywords
Failure characteristics, empirical study, J2EE, open source.

I. INTRODUCTION
J2EE application server (AS) has already become one of the most important infrastructures for enterprise software systems. It provides a set of common services, such as communication, concurrency, transaction, security and persistence, to facilitate the development, deployment, maintenance and evolution of enterprise applications. Thanks to the standardization of J2EE services and efforts of open source communities, which provide several open source/free J2EE AS products, many J2EE applications can be developed in a cost-effective way.

However, the safe-critical nature of enterprise applications makes customers of open source J2EE AS faced with a dilemma. Because on the one hand J2EE application developers and related stakeholders assume the selected server is reliable enough and some hypotheses have already indeed supported the idea [28]. On the other hand, the assumption lacks sophisticated validation by empirical data analysis (most existing validations only consider some famous Open Source Software (OSS), such as Linux and MySQL) and application developers cannot fully confirm that the selected one is reliable enough as they assumed. The cause for such dilemma of OSS’s reliability comes from, but is not limited to, the following three considerations: (1) Open source development model [37] differs from current software engineering practices and whether the new model (especially quality model) has negative or positive impacts on software qualities is under discussion [1][24][31]. (2) Open source J2EE AS often integrates many existing J2EE-compliant service implementations, which are developed by other groups or communities, to facilitate their core development. Although almost all of these services are open source, it is technically impractical to apply strict code review and other quality control methods for guaranteeing the reliability of these services. (3) A loose and unprofitable provider-consumer relationship between open source communities and their customers is not protected by contracts and laws like commercial products. Most of AS developers are unpaid volunteers and they have no duty to fix all bugs reported in a released AS as quickly as customers demand (although usually these developers are enthusiastic to fix all these bugs). We are not trying to compare pros and cons between OSS development and commercial software development, but just illustrate some reasons for the dilemma of OSS’s reliability.

Open source J2EE AS communities have realized the reliability problem. To improve the reliability of their ASs and to attract business customers to try their implementations, many open source ASs, such as JBoss [19] and JOnAS [25], use clustering to provide fault tolerance support. But the effect of clustering in open source J2EE ASs is seldom discussed before.

The study of software reliability has been active for many years and the study has already shown us a relationship between software reliability and failure. It is widely accepted that software failures reduce system’s reliability [4] and software defects account for up to 40% system failures [23]. However, these conclusions made on traditional software may be not valid any more in OSS since the circumstances of OSS development, deployment and management change more or less. As a result, some studies of the relationship between reliability and software bugs 1(a major source of failures) in OSS have been reported.

1The terms “bug” and “defect” are used interchangeably in the paper and refer to a programming fault in AS.
recently. They are different from the traditional conclusions and
the conclusions of different types of OSS are also different. In this
paper, we present a study on open source J2EE ASs with
emphasis on the following three questions:

1) How do bugs affect the reliability of open source J2EE AS?
2) Can those bug-caused failures that reduce AS’s reliability
be removed successfully if they indeed exist?
3) How can we deal with bug-caused failures that cannot be
removed or cannot be removed quickly from AS?

To answer the first two questions, we can utilize failures data
analysis, which is based on bug repository, to model reliability of
J2EE AS. Many work has already been performed to analysis
failure characteristics in Operating System (OS) ([16][20][32],
Database Management System (DBMS) ([8][13][33], Java Virtual
Machine (JVM) [10], etc. In this paper we provide a set of
measurement-based failure data analysis results of standard J2EE
services in three open source J2EE AS products, by distilling and
classifying bug reports in their bug repositories. To answer the
third question, we view the bug reports from four different
dimensions to provide some useful hints to acquire reliable J2EE
AS. In particular, we discuss how to apply software implemented
fault-tolerant mechanisms to open source J2EE AS because many
software systems have benefited from these mechanisms when
improving their reliability [11][16][18][20].

Our failure data analysis shows the following conclusions:

- A few J2EE services, such as JMS and web container, as
well as EJB container are failure-prone in all considered
ASs. Besides, different J2EE AS implementations have
different vulnerable services.
- A widely used fault-tolerant mechanism, clustering, in some
current open source ASs is not reliable enough even when
target failures (the clustering is designed to deal with these
failures) do not happen. In other words, the reliability of the
current clustering implementation itself needs improvement.
- About two-thirds of failures happened with thrown
exception or error, and about a quarter of failures happened
silently. So exception–catching is definitely a major mean
to monitor errors in J2EE AS.
- Only 15% of bug-caused failures can be removed within a
week. 74% of bug-caused failures can be removed within a
period varied from two weeks to a year. The worst thing is
that about 10% failures still exist without available patches
in a year. Most of these failures happened in EJB container,
cluster service and JMS.
- Conditions of two kinds of slippy failures caused by
insufficient CPU synchronization and improper memory
operations are not the same. Available memory debug tools
help to identity improper memory operations but more CPU
synchronization supports are needed.
- Failures related to service semantics are major threats to AS
reliability and needs more attention and efforts.

The rest of the paper is organized as follows. Section 2 gives an
introduction of bug repository in OSS, J2EE architecture and
standard J2EE services; section 3 explains our failure analysis
data source and data selection criteria; section 4 shows four
dimensions of failure analysis; section 5 presents failure analysis
results based on the former four dimensions; section 6 surveys
related work on failure analysis that is based on bug repository
discusses limits of such an approach; at last, section 7 concludes
the whole paper and identifies the future work.

2. BACKGROUND

2.1 Bug Repository in OSS

Contrary to commercial J2EE AS venders, open source
communities open its bug repository, which contains a plenty of
failure information, to all people. Both open source AS developers
and customers can issue a topic in a form of bug report when
they meet a failure during developing, testing or using. A
few experienced developers (core developers) are in charge of all
the reported bugs so that each report can be well tracked. Every
bug report includes, but is not limited to: bug summary, created
date, environment, possible source component(s), assignee(s),
description and repair comments. Bug summary field gives a
simple but refined description about bug symptoms and
description field gives a complete one, which often includes
memory dump, application code, etc. to help to reproduce the bug
during debugging. Created date field records the date when a bug
report is created at the first time. Environment field records a
hardware and/or software configurations on which the AS failed.
According to the above information, core developers give an
initial judgment about which component (service) is responsible
for the failure and who is responsible for the bug. This
information is saved in possible source component(s) field and
assignee(s) field. At last, repair comments field tracks bug fixing
history.

Failure data in our study is mainly distilled by hand from bug
summary, possible source component(s), description and repair
comments fields. In these fields, description and repair comments
are two most important sources because some bugs are so
complex or subtle that only the person who fixes them is
authoritative to its root cause and effect. A discussion about how
to fix the bug is often found in repair comments and only aided by
the discussion can we obtain a complete and precise
understanding about the bug and its corresponding failures.

A doubt about the bug repository-based approach is that whether
failure characteristics can be distilled from bug repository? We
follow the definition of fault, error and failure recognized by most
of work that a fault is an adjudged or hypothesized cause of an
error, an error is a deviation from the correct service and a failure
is an event that occurs when the delivered service deviates from
the correct service [4]. A bug is a set of software faults. Since
most researchers believe there is no way to obtain all information
about failures in software, bug or failure report analysis often
becomes the last resort. In addition to, most J2EE AS customers
are application developers, who are often familiar with J2EE
technique and can distinguish between application-caused bugs
and server-caused ones. When they find a problem in the AS, they
would prefer to report the failure and get help from AS developers
(otherwise, they can do nothing except try another J2EE product).
So, by such careful distilling, bug repository can be a suitable
source of failure description.
2.2 Services in J2EE Application Server
A J2EE AS can be regarded as a collection of common services, which provide application developers as well as applications good supports in common functionalities such as communication, transaction and security (See Figure 1). There are sixteen common services in J2EE specification 1.4 [34] and these services are the foundation of J2EE-based software systems. Table 1 shows some common J2EE services covered by our failure analysis.

![Figure 1. J2EE application server architecture](image)

Under the standardization of J2EE services, a service can be implemented by several different communities as long as the implementation is compliant to the service specification. All these J2EE-compliant services can be integrated in a platform typically using Java Management Extension (JMX) [35] or Open Service Gateway Initiative (OSGi) [27]. As a result, an open source J2EE AS has usually integrated one or more open source J2EE services. For example, JOnAS [25] integrates web services support from AXIS; Java message service from JORAM, web container from both Tomcat and Jetty, so on and so forth. Despite the benefit of reusing these free implementations, integrating services that are developed by other open source communities may introduce uncertain software defects.

3. FAILURE SOURCES
To find out failure characteristics of open source J2EE AS, we study bug repositories from three well-known open source implementations: JBoss AS from JBoss [19], JOnAS from ObjectWeb [25] and Geronimo from Apache [3]. The openness of bug repositories makes failure-related information mixed with some failure-unrelated information. So we filter bug reports according to the following criteria to expel unrelated information:

- Only released version is considered because alpha or beta versions are often not stable and many bugs in these versions had already been fixed in released versions. This means customers would be not likely to meet these failures. We get 756 bug reports from released versions of JBoss, 873 from JOnAS and 526 from Geronimo.
- Failures caused by the underlying platforms (hardware, OS, JVM or DBMS) or upper applications are not considered because they have already been studied in previous work [10][29] and we concentrate on AS itself in the study. Just as depicted in Figure 1, J2EE AS depends so heavily on the underlying JVM and OS, that failures in these platforms have an inevitable impact on the AS. At the same time, failures in a J2EE application can either affect server’s reliability. A more detailed study needs considering OS, JVM, J2EE AS and application as a whole and it would be a complex task (as our study shown in section 5.3).
- Only standard J2EE services, e.g. transaction and security, and widely used ones, e.g. clustering, are considered. Though most ASs provide some self-defined/private services, we do not consider them in order to give a uniform comparison. However, our bug repository based approach can still be applied to these self-defined services when necessary.
- Feature requests in bug repository are not included. Some reported bugs are so subtle that a few developers regard them as bugs while others believe these are requested features, which should be implemented in the future. For example, in JBoss Bug #1723 (http://jira.jboss.com/jira/browse/JBAS-1723), the bug submitter believed JACC authorization in the version is not good enough and it should deal with an unauthenticated caller more nicely. But some core developer believed that it is not dangerous and suggested changing the bug report to a feature request. Such things are not rare and some ambiguous descriptions in J2EE specifications make things worse. To get a solid result, only those bugs agreed by almost all developers are considered.
- Only bugs that would definitely cause runtime failures are considered. Some bugs are excluded because they would only lead to a mismatch between the implementation and the documents, e.g. JBoss Bug #4172 (http://jira.jboss.com/jira/browse/JBAS-4172). Other exclusion includes compile errors, e.g. JOnAS Bug #300046 (http://forge.objectweb.org/tracker/?group_id=5&atid=100005&func=detail&aid=300046), failures in test cases, e.g. JOnAS Bug #300988 (http://forge.objectweb.org/tracker/?group_id=5&atid=100005&func=detail&aid=300988), and bug reports that are not clearly described (so that we cannot decide whether these bugs could lead to runtime failures).

According to the above criteria, we check all existing bug reports in released versions, that is, 756 in JBoss, 873 in JOnAS and 526 in Geronimo, and select 256 from JBoss7, 187 from JOnAS7 and 58 from Geronimo4 (so less bug reports are selected in Geronimo because there is only one released version of Geronimo when we begin our study). Furthermore, we obtain 286 distinct failures from filtered JBoss bug reports, 192 from JOnAS and 71 from Geronimo (Sometimes, a bug report is related to more than one

---

2 URL: [http://jira.jboss.com/jira/browse/JBAS](http://jira.jboss.com/jira/browse/JBAS). We select bug reports between the day when the repository was created and November 7, 2006.

3 URL: [http://forge.objectweb.org/tracker/?group_id=5&atid=100005](http://forge.objectweb.org/tracker/?group_id=5&atid=100005). We select bug reports between the day when the repository was created and December 4, 2006.

4 URL: [http://issues.apache.org/jira/browse/GERONIMO](http://issues.apache.org/jira/browse/GERONIMO). We select bug reports between the day when the repository was created and December 6, 2006.
failure). Notice that less than one third of bug reports are selected in our study because only these bugs can result in runtime failures in released versions. The limit of such approach will be discussed in section 6.

Table 1. Some common services in J2EE

<table>
<thead>
<tr>
<th>Service name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>Aggregates a group of AS instances that together provide fail-over and replicated services to support scalable high-availability operations.</td>
</tr>
<tr>
<td>Deployment</td>
<td>Enable tools to configure and deploy applications on a J2EE platform product.</td>
</tr>
<tr>
<td>EJB Container</td>
<td>Provides a runtime space for EJBs with lifecycle management, concurrent control and coordination between EJBs and services.</td>
</tr>
<tr>
<td>J2EE Connector Architecture (JCA)</td>
<td>Provides a standard means for providing resource adapters to enable connecting to enterprise information system (EIS) resources.</td>
</tr>
<tr>
<td>Java Message Services (JMS)</td>
<td>Provides reliable asynchronous messaging implementations.</td>
</tr>
<tr>
<td>Management/Kernel</td>
<td>Usually based on a JMX or OSGi model to monitor and adjust management information of J2EE platform, application components and common services.</td>
</tr>
<tr>
<td>Persistence</td>
<td>Provides support to map object attributes to relational data.</td>
</tr>
<tr>
<td>Security</td>
<td>Ensures components and resources are accessed by only those authorized ones.</td>
</tr>
<tr>
<td>Timer</td>
<td>Enables a bean to schedule a business process to occur at a predetermined time or at a regular interval.</td>
</tr>
<tr>
<td>Transaction</td>
<td>Frees component developers from handling transactional issues.</td>
</tr>
<tr>
<td>Web Container</td>
<td>Provides a runtime space to for servlets and Java Server Pages (JSP).</td>
</tr>
</tbody>
</table>

4. CLASSIFYING FAILURES HIDED IN BUG REPOSITORY

According to the above criteria, all we finally selected bug reports contain precise and plentiful information, so that we can obtain failure descriptions such as manifestation, estimated source, severity and possible advices on fix. These descriptions can be viewed from different perspectives to get different conclusions. Because our aim is to obtain some characteristics about failures in open source J2EE AS to show the reliability status of such implementation, and more important, to obtain some guidelines or hints to improve their reliability, failure data are classified in the following four dimensions: failure source, failure manifestation, failure root cause and repair time. Failure source helps us to identify the most of failure-prone J2EE services since we believe not all services in J2EE AS have the same reliability. Failure manifestation shows the influences of different failures from users’ view. Failure root cause shows why a failure happens in a service and how developers can deal with the failure. At last, repair time is a temporal dimension which answers whether a patch for a bug is in time, which service’s bug is difficult to be fixed and which type of bug is difficult to be fixed. The analysis result according to the four dimensions helps to answer the three questions listed in the first section.

4.1 Failure source

J2EE service is a basic unit of our failure analysis and every bug report contains a field that indicates one or more J2EE services in which the failure occurred. Analysis based on failure source can tell us which service is the most vulnerable to failures in the whole server. This kind of information helps to identify some hotspots that need reliability improvement. Table 1 lists the J2EE services to be considered in the study.

4.2 Failure manifestation

Failure manifestation describes the appearance of a failure that can be seen by application developers or end users. Different failures may have different manifestations. In addition, failure manifestation is a major sign in error detection, which is the first step in fault tolerance.

There are five types of failure appearance according to our observation (as shown in Table 2). In Java programming language specification [15], a method would throw an exception, which encapsulates error information, if the method is unable to complete its task in the normal way. In some other worse situations, such as internal errors and resource exhaustion inside the Java runtime system, JVM would throw an error. Both exceptions and errors can be regarded as language level failure manifestations. Besides these two predefined types, J2EE AS would crash, fail silently or fail with non-deterministic behaviors.

Table 2. Five different failure manifestations

<table>
<thead>
<tr>
<th>Failure Manifestation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thrown exception</td>
<td>A failed service throws a predefined exception in Java programming language.</td>
</tr>
<tr>
<td>Thrown error</td>
<td>A failed service throws a predefined error in Java programming language, e.g. OutOfMemoryError.</td>
</tr>
<tr>
<td>System crash</td>
<td>System does not respond accompany printed error messages.</td>
</tr>
<tr>
<td>Silent failure</td>
<td>Includes deadlock/hang, silent crash, no response and incorrect computation result.</td>
</tr>
<tr>
<td>Random</td>
<td>A failed service shows non-deterministic manifestation.</td>
</tr>
</tbody>
</table>

4.3 Failure root cause

Failure root cause specifies which cause is the underlying cause for the observed failure, while failure source only describes where a failure happened. Failures in J2EE AS can be ascribed to three types of reasons: improper memory operation, insufficient CPU synchronization and others (most of these failures are related to J2EE service semantic) (see Table 3).

Table 3. Three different failure root causes

<table>
<thead>
<tr>
<th>Failure Root Cause</th>
<th>Description</th>
<th>Abbr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improper memory operation</td>
<td>Failures caused by improper memory operations, e.g. memory leak, dangling pointer and NULL pointer dereference.</td>
<td>Memory</td>
</tr>
<tr>
<td>Insufficient CPU synchronization</td>
<td>Failures caused by faulty programming practice or resource race.</td>
<td>CPU</td>
</tr>
</tbody>
</table>
4.4 Repair time
Application developers and related stockholders pay a lot of attention to a question that how long a reported failure could be removed after it was reported. This question becomes more concerned in OSS, just as stated in the first section. Customers of open source J2EE ASs accept the fact that bugs exist in AS, but they want to make sure whether a bug-caused failure in the AS can be removed quickly. The same question on commercial products is seldom discussed in previous work partly because this is sensitive. Emergence of OSS opens all work log and change history of a given bug. This helps us to answer the question.

We classified all selected bug reports by the meantime between the time when it was first reported and the time when it was final closed. In the life-cycle of a bug report [2], report’s status is set to “NEW” when the bug is submitted at the first time. Only after a resolution is taken and verified, the bug report becomes “CLOSED”. However, a closed bug report can be “REOPENED” when the resolution was deemed incorrect, and the cycle goes on. We take the final timestamp of “CLOSED” bug reports as its fixed time.

Noting that some recently reported bugs have not been fixed when we collect bug reports (although most of them can be fixed in the near future), we exclude them in failure repair time analysis.

5. RESULTS
In this section, we show failure analysis results according to the criteria described in section 4. In the first part, EJB container, management/kernel and JMS are pointed out because these services are failure-prone in all three J2EE ASs. Besides, cluster service is also vulnerable. In the second part, the proportions of five J2EE AS failure manifestations are shown and we find Java exception and error are the majority. In the third part, we find the proportion of three failure root causes in all three ASs are the same. At last, we answer the question that how long a bug would be fixed as usual in open source J2EE AS.

5.1 Failure source analysis
Every bug’s description and work log are checked to validate its failure source description in the report, because sometimes a specified service name in possible source component(s) field of the bug report is not the real failure source. One cause for such situation is that an error in an AS can propagate among several correlated services and a service in which a failure is manifested is not the service in which a fault exists. This complexity makes some inexperienced bug reporters unable to find the real peace breaker precisely. Figure 2(a) - (f) show the numbers of failures reported/remained in different J2EE services.

In Figure 2 (a)-(c), we can clearly find that different J2EE ASs have different vulnerable services, except EJB container, management/kernel and JMS -- these three services are failure-prone in all of the concerned ASs. In our experience, we believe the reason why these services are failure-prone lies in: EJB container provides core functionalities for J2EE application and is frequently invoked, and hence it is more likely to expose its defects; management/kernel and JMS are functional complex and hence difficult to be both implemented and debugged. Other services, such as the cluster service and web container in JBoss, the cluster service and timer service in JOnAS, and the deployment service, security and web container in Geronimo, are also vulnerable. Since all J2EE compliant ASs must obey the same set of specifications, there must be differences over architecture design or code quality in these services in different ASs. The finding also means that to augment an AS’s reliability, the efforts should be specific to a J2EE service.

Specifically, we notice that cluster service in JBoss is responsible for about 20% of total failures and the number of newly reported cluster failures in JOnAS seems on the rise. The cluster service is usually used for fault tolerance and load balance in J2EE AS. Among all failures happened in JBoss cluster service, 84% happened during ordinary operation (session replication, removing, etc.) and 16% happened when the cluster is going to be used to tolerate some anticipated failures. Almost the same proportion exists in JOnAS, 83% and 17% respectively. One of core developers from JBoss admitted that the difficulty of creating test cases for the clustering is a major cause for such condition. It seems that the developers of both JBoss and JOnAS (when we begin failure data collection, Geronimo has not cluster service support yet) should not rely only on cluster service to improve AS’s reliability.

However, someone may argue that whether a big failure number really stands for vulnerability. We admit the fact that after removing reported bugs in a service, the service becomes more reliable than before. By considering the failure data of half a year ago and that of one year ago, we believe our conclusion is valid. Because cluster service in JBoss is always buggy (see Figure 2 (a)) and cluster service in JOnAS seems not reliable enough (see Figure 2 (b)). This is not the case of a fact that some previous vulnerable services become reliable after half or one year (less failures are observed in these services), such as EJB container in JBoss, persistence service and transaction service in JOnAS as well as JCA and JMS in Geronimo (see Figure 2 (a), (b) and (c)).

Another interesting result is that failure numbers of web container in all three ASs are not the same, although all the ASs use Tomcat (JOnAS use either Tomcat or Jetty) as web container. This means different practices in different integrating groups lead to different software reliability.
Figure 2. Failure source analysis results in JBoss ((a),(d)), JOnAS ((b),(e)) and Geronimo ((c),(f)). The numbers in (a), (b) and (c) record newly reported failures in the corresponding periods (half a year), while the numbers in (d), (e) and (f) record untreated (still opened) failures at the time (every three months). We omit some services that all failures in them are removed.

Figure 2 (d)-(f) show failures that have no available patches in an AS. We account such failures in every three months (three months are considerably long for a customer who is waiting for a patch and Figure 5 (a) also showed that most of failures can be removed within three months). Less and less such failures exist in most of services in a released version of open source J2EE ASs and this implies these open source J2EE ASs are becoming more and more reliable. However, the Figure 2 (d) and (e) also support the conclusion that EJB container needs further improvement.

5.2 Failure manifestation analysis

Figure 3. Failure manifestation analysis result

Five types of failure manifestation are classified (see Figure 3) and we find that: (1) in most cases an AS would throw a Java exception or error (account for about seventy percents) when a failure happened; (2) silent failures account for about a quarter of all failures. Few failures would cause system crash or happen with non-deterministic behavior. The ratios in the three AS have some minor differences, but the trend is the same.

So, catching and analyzing Java exceptions and errors seem to be a major approach to detecting errors in J2EE AS. However, although most of thrown exceptions and errors can represent a failure faithfully, there are still a notable number of exceptions and errors that are wrongly thrown or be swallowed (altogether 49 exceptions and errors belong to this kind in our study, contrast to 267 correct ones). In addition, there are more than one-third of failures taking place without throwing an exception or error. As a result, we believe more traditional error-detect methods should be used to tolerate failures. Heart-beating, for example, can be used to detect failures that would cause an AS crash.

5.3 Failure root cause analysis

Figure 4 shows distribution of failures with three different root causes (improper memory operation, insufficient CPU synchronization and others). It is an interesting finding that the percentages of failures caused by different root causes in the three ASs are similar. There are 6%-9% failures caused by insufficient CPU synchronization, 12%-18% caused by improper memory operation, and 72%-78% caused by other programming faults. The proportions of improper memory operation and insufficient CPU synchronization are smaller than some previous study on
commercial software [33]. To our point of view, it may not imply the OSS is better in these two root causes than commercial software, instead, the OSS is worse in programming faults than commercial software.

![Figure 4. Distribution of different failure root causes](http://forge.objectweb.org/tracker/?group_id=5&atid=100005&func=detail&aid=304075);

- A new version will be released in a few days so that a patch for a reported bug will be provided only after that time (JBoss Bug #1576 (http://jira.jboss.com/jira/browse/JBAS-1576)).

The figure also shows that insufficient CPU synchronization-caused failures account for half of failures caused by improper memory operation (on average). However, the information cannot give us sufficient support to give a “big picture” of failure characteristics about the whole J2EE platform, because J2EE AS runs upon JVM and previous failure analysis results in JVM [10] showed that failures in Execution Unit account for about double of that in Memory Management Unit. Combining failure characteristics of both J2EE AS and JVM to give a holistic analysis is what users really want, but the complex dependency relationship between J2EE AS and JVM (see Figure 1) makes the analysis hard. It is still an open problem in our knowledge and needs further study.

### 5.4 Repair time analysis

Time spent to fix a bug is depicted in Figure 5 (a). It is clear that only 15% of bug-caused failures can be removed within a week, half of failures can be removed within two weeks and more than two-thirds of failures can be removed within three months. We believe the result is not very optimistic because half of failures cannot be removed within the first two weeks. Moreover, about 15% failures (in addition to failures that are still open) cannot be removed in one year. Most of these difficult-to-remove failures stay in EJB container, kernel, cluster service and JMS (not shown in the figure). The finding supports the conclusion in section 5.1 that these services are vulnerable and bug in them are difficult to be fixed.

In addition, work log in bug repository gives us some evidences that there indeed exist some bugs to be postponed due to some objective reasons. Typical objective reasons include:

- The bug-caused runtime failure is so complex that only a new algorithm or design can solve the problem (JOnAS Bug #304075 (http://forge.objectweb.org/tracker/?group_id=5&atid=100005&func=detail&aid=304075));
- Most of experienced developers are busily developing other critical modules and have no time to repair the bug (JOnAS Bug #304116 (http://forge.objectweb.org/tracker/?group_id=5&atid=100005&func=detail&aid=304116));
- Some adopted service implementations are developed by other groups and the integrating group has nothing to do with it except waiting (JOnAS Bug #303993 (http://forge.objectweb.org/tracker/?group_id=5&atid=100005&func=detail&aid=303993));

- A new version will be released in a few days so that a patch for a reported bug will be provided only after that time (JBoss Bug #1576 (http://jira.jboss.com/jira/browse/JBAS-1576)).

- Condition of improper memory operation-caused failures is better than that of insufficient CPU synchronization-caused failures. Some memory related bugs were fixed in each quarter while little improvement on insufficient CPU synchronization-caused failures is obtained after the first quarter. The difference may attribute to many modern debugging tools, such as Purify [17], that help to identify memory bugs (Some good examples are JBoss Bug #1807 and JOnAS Bug #303993, in which developers use these tools to find and fix bugs to eliminate runtime failures). But for CPU synchronization related failures, it is difficult to be reproduced and less debugging tools make condition worse. Developers face with hard problems when removing such failures. Consequently, more CPU synchronization related failures exist in AS than memory related failures after a year of maintenance, without available patches (see top row in Figure 5 (b)). In this reminder, 13% failures are caused by insufficient CPU synchronization and 4% are caused by improper memory operation. This is a contrast to the fact that more memory related failures are reported than CPU related failures (see Figure 4).

Service semantics related failure is the majority with respect to the three root causes. This kind of failures is a focus for
developers no matter in what periods. Such failures account for 68.9% of repaired bugs in the first quarter, 72.5% in the second quarter, 64.7% in the third quarter, 75% in the fourth quarter, 86.9% in more than one year and 67% in still opened failures. The above observations support the idea that memory related failures can be relative easily removed from J2EE AS, thanks to efforts of many researchers in recent years [9][38]. But CPU related failures and service semantic related failures are still two headaches for AS developers. Just as stated in [21], today’s feature-dominated development culture makes reliability low priority. Whatever, we can assure that if some temporary remedies can be provided for customers when they are waiting for a patch, they would be happier and improve the confidence on the selection of OSS. We believe there are two kinds of work to change the situation. First, more studies about synchronization programming and debugging tools could reduce bugs directly (and hence, reduce failures at runtime). Second, some approaches, such as software implemented fault tolerance, could be used to surviving remaining bugs. Many generic fault tolerance mechanisms have already been proposed in the past, such as micro-reboot, replication, checkpoint and even failure-oblivious computing [29], to deal with one or more kinds of failures.

6. RELATED WORK AND DISCUSSION

A key empirical knowledge to improve software reliability is to understand failures happening in the software. However, almost all proprietary software denied public access to their software failure information. Some studies on commercial products have already shown us some valuable observations about failure characteristics in commercial OSs and DBMSs [13][16][20][32][33][36]. These observations provide evidences for selecting one kind of fault-tolerant mechanisms, such as process pair, for the system in order to improve its reliability. But others cannot validate and repeat the work because detailed failure data is confidential and their analysis of failures seems scattered with arbitrary assumptions and extrapolations.

The emergence of OSS changed the software world. There are many free downloadable open source J2EE ASs and these are accepted by not only personal consumers but also commercial ones. According to a survey of 2004 [5], the number of companies, who use JBoss as J2EE AS, has exceeded the number of companies who use IBM’s corresponding commercial products. OSS communities open their development process. This allows us to study failures in the software. There are some work in distilling and analyzing failures in OSS [8][14]. The conclusions of these studies support or opposite to some existing assumptions, and provide some guidelines to improve the reliability of OSS. But failure characteristics of J2EE AS are never discussed to the best of our knowledge.

6.1 A review of bug repository based failure analysis

We can view previous bug repository-based failure/error/fault analysis work from two aspects: motivation and classification dimension. There are two motivations for failure analysis: (1) To test the validity of some basic software engineering hypotheses or the effectiveness of existing reliability-improvement activities [8][13][30][36]. For example, Fenton et al. [13] study fault data from two major releases of a large switching system to test some hypotheses related to: the Pareto principle of distribution of faults and failures; the use of early fault data to predict later fault and failure data; metrics for fault prediction, etc. Chandra et al. [8] use bug reports and source codes to understand and classify faults occurring in three OSS to test the hypothesis that generic recovery techniques can survive most application faults without using application-specific information. Schroeder et al [30] study failure data of a large high-performance computing system to find failure rate distribution, repair time distribution, etc. (2) To provide useful insights into failure characteristics. Lee et al. [20] study a commercial OS’s problem database and identify the failure modes of the system due to software faults. Cotroneo et al. [10] classify failure data from two JVM implementations’ bug database and get failure characteristics of JVM. These efforts are valuable for reliability improvement activities on the systems.

The classification dimensions depend on the motivation of failure analysis. For example, to test a hypothesis about the relationship between modules and failures, Fenton et al. [13] select failure number, lines of code and McCabe’s cyclomatic complexity, etc. as metrics for each module. The ultimate goal of our work is to improve the reliability of OSS using fault tolerance techniques. Therefore, we take failure source, failure manifestation, failure severity, failure root cause, environment dependency and temporal dimensions (frequency, repair time, etc.) into account. Failure source identifies in which component a failure happened and component is a basic unit to apply fault tolerance techniques. Failure severity assigns priorities to failed components in order to apply fault tolerance to the most important ones (in our study, failure severity is similar to failure manifestation). Failure manifestation provides detectable aspect of failure, which helps to detect errors in a component before the error leads to a failure. Failure root cause helps to select suitable fault tolerance techniques. For example, reboot/micro-reboot can deal with failures caused by memory leak. Combining roll back, re-execute and diversity can deal with failures caused by insufficient CPU synchronization. Diversity can be helpful to deal with environment dependent failures. At last, temporal dimensions can decide when and how often to use a selected fault tolerance technique.

6.2 The limit of failure analysis based on bug repository

There are also some limits on bug repository-based failure analysis such as qualities of bug reports, subjective administration and explanation of the reports [8][14]. One of the major limits is we cannot make sure bug repository has a capability to provide enough information for failure analysis. Most researchers already accept that bug is not the same as failure and the relationship between the two concepts is complex [4][36]. We acknowledge the complex relationship exists in our study, and the complexity has little effect on our selected failure classification dimensions. The limits also cause some inconsistencies. Some recent bug repository-based study draws discrepant conclusions, comparing to the previous work. For example, it was believed that most
residual bugs in released commercial software are Heisenbug. The reason lies in that this kind of transient bugs would cause non-deterministic failures and hence be too difficult to remove from released software [16]. However, some researchers get an opposite conclusion based on bug reports of several OSS. They find deterministic failures are dominant and non-deterministic failures account for about 9% in all bugs [8]. They attribute the change to today’s software culture that is full of rapid-changing features and codes. At the same time, another study finds that more than 38% known failures have non-deterministic behaviors [10].

In our study, we also try to find non-deterministic failures in bug repositories, but only find 16 such failures (12 from JOnAS and 4 from Geronimo). The ratio (3.8%) is lower than those in previous studies. Another surprising result is that there are not any transient failures in JBoss. Is it true? Based on findings by hand, we cannot assure transient failures are minor in open source J2EE AS because we find bug repository is not a proper data source for studying transient failures. Bug repository is mainly used to provide information for developers to fix existing bugs. Whether a report describing an unrepeatable failure is a bug report or not largely depends on the decision of administrators, who are experienced core developers, and their judgment decides the number of transient failures in bug repository. Therefore, information of transient failures in bug repository is not objective or comprehensive and solid conclusions about transient failures cannot be drawn from such information.

7. CONCLUSION AND FUTURE WORK
There are two main contributions of the paper. First, we draw several conclusions about failure characteristics of open source J2EE ASs by distilling and analyzing bug reports in three widely used open source products: JBoss, JOnAS and Geronimo. The findings help us to understand failures in open source J2EE AS. Second, we summarized other contributions on bug repository-based failure analysis and discussed the limit of such approach.

In the future, we hope to change the situation of open source J2EE AS when a bug is reported while the corresponding patch is not available in a short time. Just as stated in section 5, some bugs survive in the software for variable time in open source context. This will definitely result in software failures at runtime. Fix activities performed by developers are indeed the final and trusted solution, but how about customers when they are waiting for a patch? We believe software implemented fault tolerance can eliminate the effect and provide customers reliable software, as proposed before. Of course, the fault tolerance mechanisms in our study must be cost-effective and timesaving.

There are many fault-tolerant mechanisms can be used to deal with bugs in open source J2EE AS. For CPU-race faults, re-execute plus data diversity [12][26] can low the possibility of failures. For memory leak bugs, restart [22] (micro-reboot [6] [7]) seems to be a feasible solution except some service unavailable problems. But for service semantic related bugs, fault-tolerant mechanisms should be configured specific to the service and/or application as described before [26]. In addition, we have already proposed a coordinated fault tolerance framework in J2EE AS [22] and our future work would apply this framework to open source J2EE ASs based on the findings in this paper.

8. ACKNOWLEDGMENTS
This work is sponsored by the National Basic Research Program (973) of China under Grant No.2005CB321805, the National Natural Science Foundation of China under Grant No. 90612011 and 90412011 and the IBM University Joint Study Program.

9. REFERENCES
Transactions on Software Engineering, VOL. 26, No.8.
August 2000, 797-814.


