Characterizing Configuration Problems in Java EE Application Servers: An Empirical Study with GlassFish and JBoss

Fahad A. Arshad, Rebecca J. Krause, Saurabh Bagchi
Purdue University
West Lafayette, IN, USA
{faarshad, krauser, sbagchi}@purdue.edu

***Abstract*—We present a characterization study on configuration problems for Java EE application servers. Our study analyzes a total of 281 bug-reports in two phases: a longer (phase-1) and a shorter (phase-2) phase, from bug tracking systems of two popular open source servers, GlassFish and JBoss. We study configuration problems in four orthogonal dimensions: *problem-type, problem-time, problem-manifestation* and *problem-culprit*. A configuration problem, by type, is classified as a *paramater*, *compatibility* or a *missing-component* problem. Problem-time is classified as *pre-boot-time*, *boot-time* or *run-time*. A configuration problem manifestation is either *silent* or *non-silent*. Problem-culprit is either the *user* or the *developer* of the application server. Our analysis shows that more than one-third of all problems in each server are configuration problems. Among all configuration problems for each server in phase-1, at-least 50% of problems are paramater-based and occur at run-time. In phase-2, which focuses on specific versions over a shorter time-period, all three problem types *parameter*, *compatibility* and *missing-component* have an almost equal share. Further, on average 89% of configuration problems result in a non-silent manifestation, while 91% of them are due to mistakes by the developer and require code-modification to fix the problem. Finally, we test the robustness to configuration by injecting configuration-bugs at boot-time with SPECjEnterprise2010 application deployed in each server. JBoss performs better than GlassFish with all of the injections giving a non-silent manifestation as opposed to only 65% non-silent manifestations in GlassFish.**

***Index Terms*—configuration, Java EE, reliability**

I. Introduction

Downtime of software systems due to configuration problems is a critical issue for large-scale enterprise systems [1]. A configuration error in a software system can cause significant financial losses [2], [3]. A human mistake in entering a single character can result in a major outage; for example, a human error of misunderstanding the meaning of character "/" [4] caused all the search results returned by Google search engine to be falsely flagged as harmful for a period of up to 40 minutes in 2009. To understand the preponderance of configuration errors, consider that IT experts currently estimate that more than 80 percent of network outages occur due to configuration errors [5]. Arguably network configuration management is a challenging task, but it is safe to extrapolate and conclude that a significant fraction of other failures are also caused by configuration errors.

To minimize these losses, software systems should be built in a manner that is resilient to configuration errors. Further, to improve resilience, the testing of the developed software is a critical phase for all software products. Unfortunately, due to time-to-market constraints, artificial testing environments and insufficient configuration testing, bugs related to configuration problems often are pushed to users of the software.

To make matters worse, it is not clear who is responsible for a given configuration problem. Developers blame users of incorrect configuration at the user-end, while users blame the software developers. This blame game of each other leads to longer bug-resolution time. Though, user-training helps to minimize the occurrence of configuration problems, developers are expected to play a more pro-active role from the perspective of developing robust and resilient software.

In order to improve software resilience towards configuration bugs, we present a characterization-study of configuration problems[1] in Java EE (Enterprise Edition) based application servers, GlassFish [6] and JBoss[2] [7]. Jave EE application servers are complex with large number of configuration parameters. In addition, as they become an integral part of cloud-based platforms [8], both their configuration complexity and their importance are only bound to increase.

We classify configuration-bugs according to four dimensions: *Type, Time, Manifestation* and *Responsilbility*. The first dimension (*Type*) considers a configuration-bug due to a problem relating to a particular parameter option, incompatibility with an external library or a missing component. The second dimension (*Time*) refers to the timepoint at which the configuration-bug is triggered, i.e., pre-boot-time, boot-time, or run-time. The third dimension (*Manifestation*) refers to whether a bug is silent or non-silent (with a log message). The fourth dimension refers to who (developer or user) is responsible for a given type of configuration bug.

Based on the characterization, we built a tool called `ConfInject`, that injects configuration errors (misconfigurations) derived from bug databases. A fault-injection tool like `ConfInject` will help to measure the robustness, the ability to handle unintended input, and can play an important role in

[1]In this paper, we use the terms problem, issue, error, and failure interchangeably.

[2]We use the more common name, JBoss application server, as opposed to its recent new name WildFly

the decision making process of choosing a Java EE vendor. `ConfInject` follows a similar design-flow as `ConfErr` [9], a tool that injects human-induced configuration errors and measures the reaction of the system under test (SUT). `ConfErr` injects configuration errors derived from human psychology rather than being grounded in real bugs that have been observed in real software. Further, it only covers boot-time bugs, one point in one of four dimensions that we consider.

As opposed to an exhaustive parameter sweep, that can take a large amount of time, `ConfInject` selectively injects a relatively small set of problems that are encountered in bug-repositories. This avoids the search-space explosion problem and reduces the running time of `ConfInject`, thus making it an effective testing tool. `ConfInject` injects character-strings in configuration data values that are stored as xml-attribute values. It then programmatically starts the application server, deploys an application, stops the server and analyzes the reaction to injection from the collected server logs. Ideally, a given injection should be manifested at boot-time (as oppose to waiting until a web user reports it at runtime) thus providing a direct feedback to the operator about a problem. Our experiments show that some injections result in silent manifestations after boot-time, thus requiring more intrusive configuration validation tests for Java EE servers.

`ConfInject` specifically injects addition and removal of the forward-slash ("/") character and replacement by empty-string (""). The injections are done across both the application and the application server configuration values in GlassFish and JBoss. The results show that as high as 35% of injections result in silent failures in GlassFish while all injections result in non-silent manifestation in JBoss.

We claim the following contributions in this work:

- Characterization of configuration-problems for Java EE application servers, where we categorize real configuration problems across four dimensions, *Type*, *Trigger-Time*, *Manifestation* and *Responsible-party*.
- Fault-Injector for configuration-problems in Java EE application servers, where a configuration-resilience profile in terms of percentage of non-silent manifestations is provided for the two most widely used Java EE application servers.
- Recommendations for better configuration management where we provide suggestions for developers to avoid such mistakes.

The rest of this paper is organized as follows: Section II presents the threats to validity of this study. Section III presents the organization of Java EE application servers. Section IV covers the classification of configuration problems. Section V explains the classification results. Fault-Injector design and results are presented in Section VI and Section VII. Section VIII discusses important observations and recommendations for better configuration management. Related work is presented in Section IX. The conclusion and future work is presented in Section X.

II. Threats to Validity

All characteristic studies have limitations; for example, the number and type of bug-reports studied can be non-representative from the point-of-view of the studied problem. Studying bug-reports and classifying them is a manual process and can be subjective.

Each bug-report can state the problem with varying levels of detail for reproducing a given bug. The quality of a bug-report often depends on the expertise of the user submitting the bug-report. We do not have any way of characterizing the expertise of the user who submits a bug report. A bug report that does not have enough information to be classified as configuration-error is classified as non-configuration error. Also, an expert user might not report a given configuration problem as opposed to a novice user. Therefore, we argue that the statistics reported in this work are a lower-bound and in reality, configuration bugs likely occur with higher frequencies.

III. Java EE Application Server Design

Here we describe the design principles and implementation aspects of Java EE application servers that are germane to understanding the incidence of configuration errors in them. A Java EE application server provides tools and APIs to develop multi-tier enterprise applications. It is designed to implement several Java EE standards represented by Java Specification Requests (JSRs) [10]. Therefore, an application server implements server-generic design principles to provide different functionalities, for example, a module that implements security functionality. Common modules in an application server include, web-container, ejb-container, persistence-management, command-line, GUI-based administration, messaging (JMS) etc. The functionality implemented by each module in a particular Java EE application server can vary. For example, vendor A can implement ejb-container functionality in one module, while vendor B might decide to implement it by a co-ordination of several modules.

Figure 1 presents basic modules and interfaces for a Java EE application server from the perspective of configuration. The arrow-heads in Figure 1 point to server components that require configuration data from outside. At *boot-time*, JVM parameter configurations are configured for a Java EE server. At *run-time*, an administrator can create server resources such as JDBC resources (connection pools), JMS (Java Messaging Service) resources, etc. These resources provide applications with a way to connect to other components, JDBC for database or JMS for asynchronous services (e.g., mail services). The creation and destruction of these resources, a configuration task, is executed either by using the commmand-line interface or administration-console in the web browser. Both command-line and admin-console utilities, which come bundled with all major Java EE servers, are used to execute several other configuration tasks as well, such as deployment of a web (`*.war`) or an enterprise (`*.ear`) application.

The components of GlassFish encountered in the studied bug-reports are: *admin, admin_gui, build_system,*

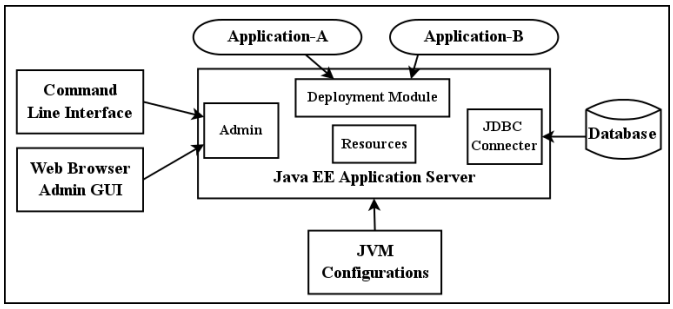

Figure. 1: Java EE Server Components Interacting with Configuration Data

command_line_interface, entity-persistence, grizzly-kernel, installation, jdbc, jms, OSGi, packaging, rest-interface, update_center, web_container, web_services and *web_services_mgmt*. *admin* is the core administration infrastructure component that handles all the configuration data entered by either the *command_line_interface* or *admin_gui*. JBoss implements its server components at a coarser granularity when compared to GlassFish. The components from studied bug-reports include: *Build System, CMP service, EJB2, JMS (JBossMQ), Other, Test Suite, Web Services, Web (Tomcat) service* and *Weld/CDI*.

IV. Classification of Configuration Problems

To characterize configuration errors, we study configuration issues in *four orthogonal dimensions: Type, Trigger-time, Manifestation, and Responsible-party*.

A. Type of errors in Java EE application servers

We are first going to define configuration and non-configuration bugs and then eliminate non-configuration bugs from further discussion.

1) Non-configuration: This is an error that results due to an interaction of internal components, without directly operating on data input by a human, either user or administrator. Examples of this include an incorrect returned object by a method, an incorrect method being called, a missing method call, an incorrect `cast` operation, etc. This type of error can manifest as failures called `Exceptions` in Java programs. Typical examples of `Exceptions` thrown at run-time due to non-configuration problems are `ClassCastException`, `ArithmeticException` etc. Other exceptions like `NullPointerException` can be thrown irrespective of whether the problem origin is non-configuration or configuration.

2) Configuration: This is a problem that is a result of operating on data that a human user configures in a file, in an administration console, in an environment variable or via command-line interface. The configuration errors can further be of the following types:

a) Parameter: This a problem that occurs due to a wrong parameter type, value or format. Examples include an empty tag value in an xml tag (JBAS-929), an incorrect value passed by the user (GLASSFISH-17581), a missing forward slash "/" in resource path (JBAS-1115), etc.

b) Compatibility: This is a problem that occurs due to an incompatibility with the environment, e.g., the JVM or a library. Examples of this type of configuration-problem are: JBAS-5275, where while reading xml-attribute values, the application server reads the values in different orders when run in different JVMs. The application server handles the values for attributes when read in forward direction, but fails to handle when read in backward direction for some JVMs. GLASSFISH-17764, where starting the cluster fails when started on the JRocket [11] JVM only. GLASSFISH-14922 where the enabling of the secure mode of the server (`enable-secure-admin`) fails on an old JVM.

c) Misplaced-Component: A misplaced-component (henceforth called component only) is a type of configuration problem which occurs when a component or a file is missing or in the wrong location and subsequently causes an application to fail. Typically, a `ClassNotFoundException` or `NullPointerException` is thrown as a manifestation. Examples of this type of configuration-problem are: GLASSFISH-17462, where a missing file (`ssl.json`) manifests as a `NullPointerException` in server log, while the user is shown a HTTP 404 error screen in administration console. GLASSFISH-17649, where while configuring the server, `java.lang.NoClassDefFoundError` is manifested due to a missing classpath dependency. AS7-1719, where `ClassNotFoundException` is thrown when the server creates a lot of http sessions within a short time interval.

B. When does the configuration error occur?

1) Pre-Boot-time: If a Java EE application server or an application fails to compile or allocate pre-boot resources due to any type of a configuration problem, it is considered to occur at *pre-boot-time*.

2) Boot-time: If a configuration error is observed at application server startup or application deployment, then it is classified as a boot-time failure due to a configuration problem.

3) Run-time: An error that affects a running application on an application server while users are accessing the application is a *run-time* problem. Configuration problems that occur after an application is successfully deployed on the server fall in this category. Arguably this is the most serious of the three kinds because a user, and a potentially non-expert one, is affected by this.

C. In what manner do the configuration errors manifest?

1) Silent: A silent error can occur as a performance problem, data-corruption or a problem that is not directly detectable by the operator and is not manifested as a relevant `Exception` in the server logs. An example of silent error is GLASSFISH-18875, where the deployment (a configuration action) of an `ear` archive is very slow and upon fixing, the deployment-time reduces from 50 minutes to 2 minutes only.

2) Non-Silent: A non-silent error occurs with a clear manifestation, typically as `Exceptions` in the server logs. The silent errors are the more serious kind, due to the difficulty of detecting them.

D. Who is responsible?

We consider a configuration problem from two different perspectives of who inserted it — developer and user. A configuration bug from the user point-of view may not be a bug from developers point-of-view, for example, if the user forgot to set the value for a paramater "`host`", it is a configuration problem of the user. On the other hand, if the user set the value for "`host`" correctly, but the value was not correctly interpreted (resolving hostname to IP incorrectly etc.), then it is a bug by the developer and requires a code change. So, somewhat simplistically, we come up with the following definition of the classes.

1) Developer: A configuration bug that is fixed by a code change made by the developer

2) User: A bug that is fixed by making the appropriate change in the configuration file, menu. etc.

V. Classification Methodolgy and Results

A. Methodology

We conduct our characterization study as two separate sub-studies, which we will call Study-1 and Study-2 (Table I). In Study-1, we constrain the bug space from all bugs (GlassFish=15805, JBoss=4160) to bugs that have the state 'Fixed' for the attribute `RESOLUTION` and the state 'Resolved' or 'Closed' for the attribute `STATUS` in the time intervals [23-May-2005,16-Mar-2012] (GlassFish) and [11-Apr-2001,11-Mar-2012] (JBoss). This represents 64% of all GlassFish and 56% of all JBoss bugs. From this set, we uniformly sampled 1% of bugs giving us a total of 124 bug reports (101 (GlassFish) and 23 (JBoss)) for manual analysis.

In Study-2, our goal is to focus in more on configuration bugs pushing automated queries to bug databases as much as we can. We constrain our search query time intervals to target one specific version of GlassFish (ver.3.1.2 - all 23 builds within it) and one specific version of JBoss (ver.7 - all 5 builds within it). We create a keyword-based search to find configuration bugs (Listing 1). This search query looks for keywords indicating configuration problems present in both the description and the comments section of the bug report. We believe that this will provide bug reports that indicate a mutual agreement (whether an issue is a configuration or non-configuration) between users and developers; developers typically put comments about the bug type in the comments section. This search query results in a total of 157 bug reports (132 (GlassFish) and 25 (JBoss)) that are now manually inspected in Study-2.

```
project = GLASSFISH AND (summary ~ "config* || setting* ||
     setup || set-up || set up" OR description ~ "config* ||
      setting* || setup || set-up || set up") AND issuetype
     = Bug AND (resolution= Fixed OR resolution = Complete)
     AND CREATED >= "2011/08/03" and CREATED < "2012/07/17"
     AND (comment ~ "config* || setting* || setup || set-up
     || set up") ORDER BY created ASC, key DESC
```

		# Issues Studied	Time-interval	Versions
GlassFish	Study-1	101	05/23/05–03/16/12	beginning till 3.1.2
	Study-2	132	08/03/11–07/12/12	3.1.2 (23 builds)
JBoss	Study-1	23	04/11/01–03/11/12	3, 4, 5, 6
	Study-2	25	11/01/10–09/21/12	7 (5 builds)

TABLE I: Statistics of bug reports that form our studies

```
project = AS7 AND (summary ~ "config* || setting* || setup
     || set-up || set up" OR description ~ "config* ||
     setting* || setup || set-up || set up") AND issuetype =
      Bug  AND (resolution= Done) AND (status= Resolved or
     status=Closed)  AND created >= "2010/11/01" AND created
      < "2012/09/21" AND (comment ~ "config* || setting* ||
     setup || set-up || set up") ORDER BY created ASC, key
     DESC
```

Listing 1: GlassFish and JBoss keyword-based search queries for automatically zooming into configuration bugs (with imperfect success)

B. Study-1 Results

In this subsection, we present the results of our analysis for Study-1, a total of 124 bug reports, from GlassFish and JBoss application server bug repositories [12], [13]. Table II shows the distribution of configuration and non-configuration problems. It is determined through manual analysis that more than one-third of problems in each server are configuration problems. This provides quantitative evidence for our claim that configuration errors are a significant source of concern.

	GlassFish	JBoss
Configuration	33%	43%
Non-Configuration	67%	57%

TABLE II: Study-1: Distribution of Configuration and Non-Configuration Problems

We observe that for GlassFish, nine components have more than 40% of their issues classified as configuration-problems. Among these nine components, the top two components having the most number of configuration problems are `admin` (6/14 (43%)), `admin_gui` (10/18 (56%)). This is an expected result as `admin` and `admin_gui` components in GlassFish are responsible for processing the configuration data passed in by the user. This pattern is not very visible for JBoss, mainly because of the few number (twenty-three) of bugs analyzed across many (nine) components.

1) Distribution by Classification Category: We present the percentage frequency distribution of configuration problems for GlassFish and JBoss in Figure 2 and Figure 3 respectively. For GlassFish, we observe a majority of the problems (79%) relate to configuration parameters, while 70% of the time, the configuration problem occurs at run-time. Additionally, 91% of all configuration problems are non-silent and 91% of the

problems are attributed to the developer. For JBoss, only 40% of configuration problems are classified as parameter-based, while 50% are due to misplaced-components. In addition, there are no issues that occur silently, a fact which is also verified by our injection tool (Section VI).

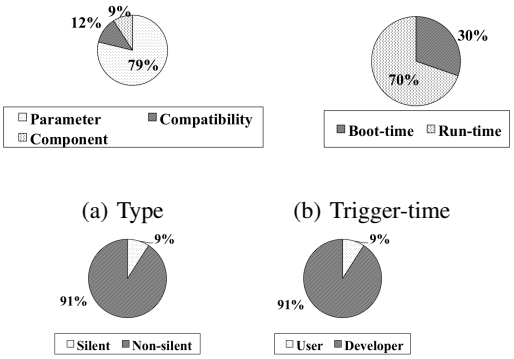

Figure. 2: Study-1 GlassFish: Frequency Distribution of Configuration Problems in each Dimension (Total = 101 bugs)

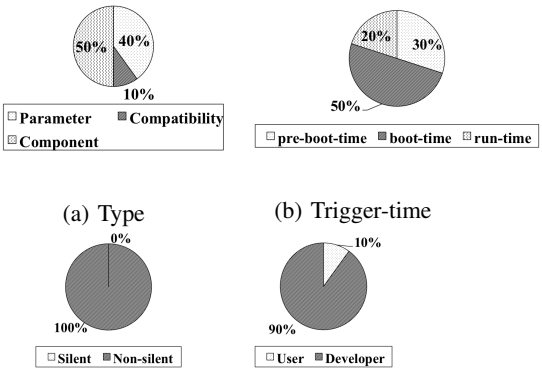

Figure. 3: Phase 1 JBoss: Frequency Distribution of Configuration Problems in each Dimension (Total = 23 bugs)

C. Study-2 Results

In this subsection, we present the results of our analysis for Study-2, a total of 157 bug reports from GlassFish and JBoss. We observe that our search (Listing 1) did help to find more configuration problems because we observe an increase in configuration problems from study-1 to phase-2 (Table III). Specifically, 33% to 62% for GlassFish and from 43% to 56% for JBoss.

1) Distribution by Classification Category: The percentage frequency distribution within each configuration problem dimension is presented in Figure 4 and Figure 5 for GlassFish and JBoss respectively. For Type (Figure 4.(a)), the parameter-based configuration problems for GlassFish decreases from

	GlassFish	JBoss
Configuration	62%	56%
Non-Configuration	38%	44%

TABLE III: Study-2: Distribution of Configuration and Non-Configuration Problems

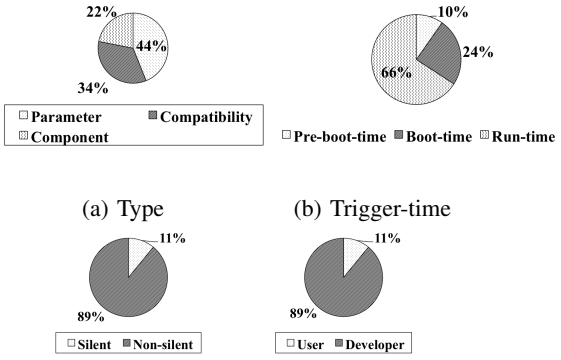

Figure. 4: Study-2 GlassFish: Frequency Distribution of Configuration Problems in each Dimension (Total = 132 bugs)

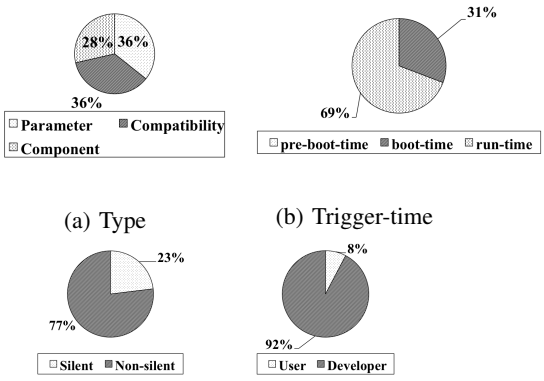

Figure. 5: Study-2 JBoss: Frequency Distribution of Configuration Problems in each Dimension (Total = 25 bugs)

79% in study-1 to 44% in study-2, while compatibility issues increase from 12% to 34%. One reason for this is that study-1 analyzes issues across various versions of GlassFish. Parameter names, values and format are more likely to change across versions (as opposed to within versions), thus giving a greater probability to see parameter-related issues for study-1 as opposed to study-2 (which focuses on one specific version of GlassFish). The distribution by trigger-time (Figure 4.(b)) is similar to study-1, where run-time configuration problems have the majority (66%) followed by boot-time (24%) problems. Also the patterns for problem-manifestation and responsible-party in GlassFish are similar to study-1.

For JBoss in study-2, we observe that all three subtypes, parameter, compatibility and misplaced-components have al-

most one-third share (Figure 5.(a)) of the configuration problems. This pattern is different from study-1's results, where parameter-based problems were dominant (50%) followed by misplaced-components (40%). The primary reason for this is that study-1 studied JBoss application server versions 3,4,5 and 6 which were very similar in their design, whereas study-2 studied version 7, which provided the same functionalities as previous versions, but had major structural differences and even required a migration guide [14] for migration to version 7 from previous versions. One well-known side-effect of this refactoring is that problems related to compatibility will happen more. This is confirmed by our analysis (an increase from 10% to 36%). For problem trigger-time in study-2, more run-time (69%) problems occur as opposed to more boot-time (50% problems in study-1). This could also be attributed to the major refactoring of JBoss version 7. The patterns for the last two studied catogories, *manifestation* and *responsible-party*, are similar for both study-2 and study-1 in JBoss.

VI. Fault Injector

Based on insights from our characterization, we developed a configuration bug injection tool called `ConfInject`. We believe that running `ConfInject` with any given Java EE application server would give the user an opportunity to assess the configuration-resilience of the server. On the other hand, this can help organizations to evaluate a given application server before deploying it in a large-scale in the production environment.

`ConfInject` follows a workflow as shown in Figure 6. Given a Java EE application server installation location and an application that can run on it, `ConfInject` emulates normal server-management workflow, i.e., starts the server, deploys an application, adds and deletes server resources, and then stops the server. While executing this workflow, `ConfInject` injects configuration problems. Currently `ConfInject` injects parameter-based configuration problems at boot-time.

A fault injector should know the answers to the following questions:

What to inject? `ConfInject` injects character-string-based misconfigurations that were observed while conducting our characterization study. For this paper, we choose to inject the character "/" and empty-string (""). Forward-slash character has been the cause of several parameter-related configuration problems in Java EE application servers [15] (JBoss), [16] (Geronimo),[17] [18] (GlassFish), [19] (TomEE) as well as in general Java programs [20] (RestEasy), [21] (OpenSSO), [22] (SailFin), [23] (JavaServerFaces) etc. configuration problems related to empty-string for server attribute-values have also been reported in both GlassFish [24] and JBoss [25].

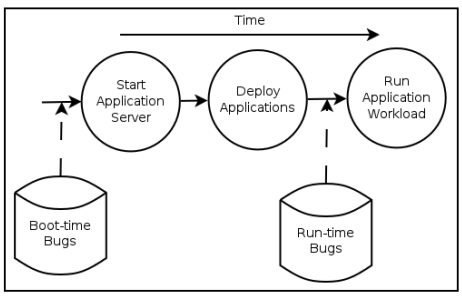

Figure. 6: WorkFlow for Time of Injection

Where to inject? `ConfInject` injects its misconfigurations in the `xml` configuration files that are maintained by the Java EE application servers for configuration management. Specifically, it injects in `xml-attribute` values within `domain.xml` (GlassFish) or `standalone-full.xml` (JBoss), `web.xml`, and `persistence.xml` files. `domain.xml` in GlassFish and its equivalent `standalone-full.xml` in JBoss maintain the major server configurations that are read during the boot-up process. The last two are configuration parameters of the application (SpecJEnterprise2010 in our case). `web.xml` is used to configure the application server resources used by a given deployed application, e.g., how URLs map to servlets, authentication information etc. `persistence.xml` is used to configure settings for the connection to back-end database, e.g, the driver-name, persistence-provider-name, etc.

When to inject? `ConfInject` injects its misconfigurations at boot-time. Boot-time can be either application-server start or an application deploy operation.

How to inject? `ConfInject` injects its boot-time misconfigurations by parsing the `xml` configuration files, mutating only one attribute-value at a time and writing the mutated file before the application-server or the application is started. For each injected string, the application server is programatically (using CARGO API [26]) started, an application is deployed, the application server is stopped and the server log file is saved for analysis. This process is repeated for all injections. We define three injection-operators, *Add*, *Remove* and *Replace*. *Add* means that we append the injected character to a similar, already existing character in the attribute-value-string. *Remove* means that we remove the injected character from the correct attribute-value-string. *Replace* means that we replace the correct attribute-value with a different string, i.e., either "/" or the empty string. These operators were defined based on evidence from bug repositories [18], [15], [24]. An example of a sample injection for each of the *Add*, *Remove* and *Replace* operating on `domain.xml` is shown in Table IV.

Target Application: SPECjEnterprise2010 We need an application running on the Java EE application servers to stress it with a realistic workload. We deploy SPECjEnterprise2010 [27] on GlassFish and JBoss.

SPECjEnterprise2010 is an industry-standard Java Enterprise Edition (EE) benchmark that emulates a complete enterprise system. It describes an end-to-end business process and emulates an automobile dealership, manufacturing, supply chain management and order/inventory system. The set of functionalities that this benchmark includes are: *Dynamic Web page generation, Web Service based interactions, Transactional and Distributed components, Messaging and asynchronous task management and Multiple company service*

Mutation Operator	Original Value	Mutated-Value
Add	<jdbc-resource jndi-name="jdbc/__default" pool-name="DerbyPool"/>	<jdbc-resource jndi-name="jdbc//__default" pool-name="DerbyPool"/>
Remove	<jdbc-resource jndi-name="jdbc/__default" pool-name="DerbyPool"/>	<jdbc-resource jndi-name="jdbc__default" pool-name="DerbyPool"/>
Replace	<property name="URL" value="jdbc:mysql://hostname:3306/specdb"/>	<property name="URL" value=""/>

TABLE IV: An example of mutated configuration values in `domain.xml` in GlassFish

providers with multi-site servers. The system architecture corresponds to a 3-tier architecture, i.e., client, middleware and database.

The clients of SPECjEnterprise2010 are automobile dealers, who use a web based user interface to access the application. The interface allows customers (car dealers) to login to their accounts, keep track of dealership inventory, sell automobiles, manage a shopping cart and purchase automobiles.

VII. Fault Injection Results

For one injection, `ConfInject` mutates one string in one of the xml files, starts the application server, deploys the SPECjEnterprise2010 application, stops the application server and collects the server logs for analyzing the effect of injection. The number of non-silent manifestations for boot-time injections are presented in Table V.

For a total of 132 injections of "/" character in GlassFish, both the server configurations in `domain.xml` and the application configurations in `web.xml` have silent failures for the majority of injections. Specifically, only 24.2% of remove-operator-based injections and 22% of add-operator-based injections result in non-silent manifestations with `Exceptions` thrown in server log file. On the other hand, all the JBoss injections result in non-silent manifestations. For empty-string ("") injections, 75.9% in GlassFish and 100% in JBoss are non-silent. These results mean that in GlassFish, the throwing of `Exceptions` is delayed until the end-users (users of SPECjEnterprise2010) try to access a particular resource. As an operator of the application server and applications, this would result in a higher frequency of issues filed by the users, a fact that is verified by observing (Figure 7) the total number of issues filed for each of the projects. We see qualitatively that the number of configuration-related issues follow a similar pattern.

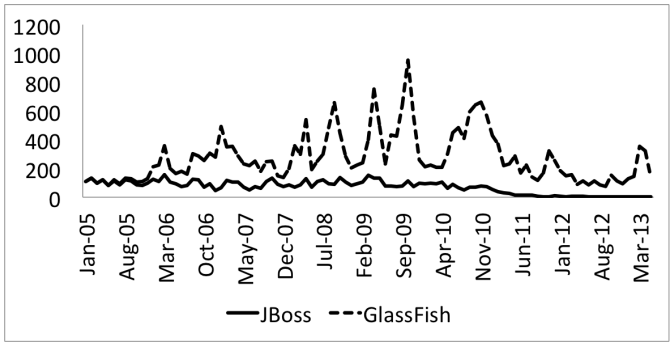

Figure. 7: Number of issues filed to GlassFish and JBoss (vers. 3-6) bug repositories per month

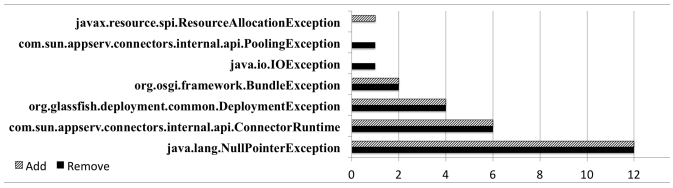

Figure. 8: Frequency Distribution of Exceptions for non-silent manifestations for GlassFish in `domain.xml` with "/" injection

A. GlassFish Injection Results

1) Forward-slash Injection: For `domain.xml` in GlassFish, the non-silent `Exceptions` distribution when forward-slash is injected is presented in Figure 8. It shows that `NullPointerException` occurs most frequently. For these `NullPointerExceptions`, the root-cause is due to the injection at two seperate locations related to JMS configurations: (1). Injection in the attribute `pool-name` within xml-element `connector-resource`, where the pool-name is changed from "jms/value" to "jms//value", (2). Injection in the attribute `name` within the xml-element `connector-connection-pool`, where the name is changed from "jms/value" to "jms//value". Both of these attribute values should match for the correct operation, thus avoiding `NullPointerException`. Given that the standard way to delimit is using a single "/", we recommend a deployment verifier that would verify and favorably replace the extra forward-slash introduced, due the configuration error thus automatically fixing the issue. Note that GlassFish does have a verifier tool called `verify-domain-xml`, which is claimed to be much more than an XML syntax verifier. Its documentation [28] states: "Rules and interdependencies between various elements in the deployment descriptors are verified". We tested it with the injected error in `domain.xml`, and it did not catch the injected forward-slash configuration problem.

For most of the injections (94.3% with remove and 100% with add operator) in `web.xml`, the server.log did not show any `Exceptions` at boot-time. This fact where exceptions are delayed until the end-user accesses the application through the web browser is a point of concern.

```
<servlet>
        <servlet-name>purchase</servlet-name>
        <jsp-file>//purchase.jsp</jsp-file>
</servlet>
<servlet>
        <servlet-name>shoppingcart</servlet-name>
        <jsp-file>shoppingcart.jsp</jsp-file>
</servlet>
```

GlassFish						JBoss			
		Application		Server		Application		Server	
Operator	Injected character	web.xml	persistence.xml	domain.xml	Total	web.xml	persistence.xml	standalone-full.xml	Total
remove	/	2/35 (5.7 %)	4/4 (100 %)	26/93 (28.0 %)	32/132 (24.2 %)	35/35 (100 %)	4/4 (100 %)	15/15 (100.0 %)	54/54 (100 %)
add	/	0/35 (0 %)	4/4 (100 %)	25/93 (26.9 %)	29/132 (22.0 %)	35/35 (100 %)	4/4 (100 %)	15/15 (100.0 %)	54/54 (100 %)
replace	""	-	-	460/606 (75.9 %)	460/606 (75.9 %)	-	-	397/397 (100 %)	397/397 (100 %)

TABLE V: Number of non-silent manifestations for boot-time injections in GlassFish and JBoss

File	Remove			Add		
	Exception	#	Percentage	Exception	#	Percentage
standart-full.xml	org.jboss.msc.service.StartException	11	73	java.lang.NullPointerException	1	6.6
	org.jboss.as.controller.persistence.ConfigurationPersistenceException	4	27	org.jboss.msc.service.StartException	13	87
				java.lang.IllegalArgumentException	1	6.6
web.xml	java.lang.IllegalArgumentException	16	46	org.jboss.msc.service.StartException	35	100
	org.jboss.msc.service.StartException	19	54			
persistence.xml	org.jboss.msc.service.StartException	4	100	org.jboss.msc.service.StartException	4	100

TABLE VI: Detailed Exception Distriubtion in JBoss when "/" was Injected

Listing 2: Two sample injections (underlined) using *add* and *remove* operators in `web.xml` that were not detected at boot-time in GlassFish

2) Empty-String Injection: Out of 606 empty-string injections in `domain.xml` within GlassFish, 69.5% (421) of them resulted in the failure of server boot-up process while 6.4% (39) resulted in `Exceptions` in server log. This means that there were still a significant number (146 (24.1%)) of empty-string-based problems that were silent at boot-time.

B. JBoss Injection Results

1) Forward-slash Injection: For JBoss, we observe in Table VI that for all injected file types and both mutation operators `org.jboss.msc.service.StartException` is thrown 80% (86/108) of the time. The `StartException` is a JBoss-defined `Exception` that is thrown when a service fails to start. This high-level exception in JBoss provides enough details in server log to diagnose the root-cause (injection location). As an example, consider the effect of applying the *remove* operator in the JMS queue entry name to change it from `"java:jboss/jms/LoaderQueue"` to `"java:jbossjms/LoaderQueue"` in `standalone-full.xml`. The corresponding server log entry is shown in Listing 3, where the corresponding problem (missing forward-slash) is observable (manifestation underlined in Listing 3). Another example with injection in `web.xml` is the second injection in Listing 2, which GlassFish did not detect whereas JBoss gave a clear manifestation as shown underlined in Listing 4. These are desirable manifestations and show that JBoss is robust enough to diagnose the injected configuration problems, both at the application and the server level. After diagnosis, a subsequent automated replacement or deletion (in the case of an extra character) of the forward-slash is still a desirable but missing feature.

```
15:58:44,916 ERROR [org.jboss.msc.service.fail] (
    ServerService Thread Pool -- 58) MSC00001: Failed to
    start service jboss.messaging.default.jms.queue.
    LoaderQueue: org.jboss.msc.service.StartException in
    service jboss.messaging.default.jms.queue.LoaderQueue:
    JBAS011639: Failed to create queue
Caused by: java.lang.RuntimeException: JBAS011846:
    Illegal context in name: java:jbossjms/LoaderQueue
```

Listing 3: Log entry after applying remove-operator on "java:jboss/jms/LoaderQueue" to remove the forward-slash

```
15:07:43,301 ERROR [org.apache.catalina.core] (
    ServerService Thread Pool -- 64) JBWEB001097: Error
    starting context /specj-specj: java.lang.
    IllegalArgumentException: JBWEB000273:
    JSP file shoppingcart.jsp must start with a "/"
```

Listing 4: Log entry after applying remove-operator on "/shoppingcart" to remove first forward-slash

2) Empty-String Injection: All empty-string injections in `standalone-full.xml` within JBoss result in the failure of server boot-up process. The corresponding log entry, `"Server boot has failed in an unrecoverable manner"`, shows that all misconfigurations need to be fixed before a successful boot-up operation, a desirable outcome.

VIII. Discussion

In this section, we highlight some of the high-level observations from our study and recommendations for improving the management of Java EE software systems.

A. Observations

1) Our observations reveal that more than one-third of problems reported by users are configuration-related. These problems occur more in components that are at the interface between the user and the program.
2) Inter-version and intra-version configuration problems tend to have different characteristics. Inter-version problems are majorly parameter-related while intra-version

problems have an almost equal share for each of *parameter*, *compatibility* and *missing-component* related issues.
3) Code refactoring or re-implementation results in an increased number of *compatibility* issues.
4) Parameter-based fault-injection, as done in our tool `ConfInject`, based on bug reports, can provide a measure of robustness of the software.

B. Recommendations for better configuration management

1) **Automated Fixing of Parameter Values:** Today, incorrect parameter values specified by users of application servers are fixed by users themselves. They primarily go through the manual process of *searching* for similar problems, *communicating* with developers, *trial-and-error* of different possible values and even *applying* patches. We suggest to developers of application servers to take a more pro-active role and dynamically fix configuration mistakes that users make. For example, given that a file name cannot contain the character "/" and many configurations have a well-defined format, e.g., token seperation of single "/" character, any extra "/" characters should automatically be removed at boot-time with a notice to the user. This process requires configuration validation, a step that is not mandatory according to Java specifications and hence not pursued by many vendors. To strongly enforce it, there is a need to push for this in the future versions of Java EE standards.

2) **Efficient Bug Repository Maintainence:** Operating a bug repository where it is easier for users to search and fix problems helps to reduce bug-resolution time and frequency of issues. From our experience while studying bug-reports in GlassFish and JBoss, we observed some desirable features in the bug-tracking systems which we recommend: (1). JBoss employs a duplicate bug-detection tool called `Suggestimate` [29]. Suggestimate clusters similar issues together. Also, as users are typing in their bug-reports, similar issues based on the keywords used are presented. This helps to reduce average number of issues filed by users. (2). For a majority of fixed bugs, JBoss cross-references the bug-fixes giving an option for the developers and users to view the difference to previous version, thus providing an opportunity to better understand the fix.

IX. Related Work

From the perspective of configuration management, software systems reliability has been studied from different angles, characterization, testing, detection, diagnosis and recovery: (1). Failure characterization [30], where the goal is to study the cause and effect of failures to identify misconfiguration-patterns. (2). Resilience testing [9], where the goal is to study the reaction of a software system when injected with configuration errors to identify weaknesses in failure handling mechanisms. (3). Configuration error-detection [31], where the goal is to study failure manifestations of misconfigurations to identify patterns that detect a given configuration-error. (4). Failure diagnosis [32], where the goal is to find a root-cause of a given configuration problem. (5). Failure Recovery [33], where the goal is to find the candidate fixes for a configuration problem.

Failure characterization studies using bug-reports, both on configuration [30] and non-configuration [34] bugs, reveal common patterns that lead to failures. Yin *et al*. [30] studied user forums for five non-Java EE systems from the perspective of configuration errors. Some dimensions studied in their work, e.g. *type* and *manifestation* of configuration problem are similar to our work, while others e.g. *responsible-party* and *problem-time* are different. Another major difference is that their work focuses on non-Java EE systems, whereas we claim that Java EE based systems have distinctive differences (e.g. standardized APIs, modular architecture) that warrant separate attention. Also, as cloud-based Java EE system like OpenShift [8] and GlassFish 4 become mainstream, Java EE based systems would require more focused attention towards configuration-related issues.

Keller *et al*. [9] tested the configuration system of five non-Java EE systems by injecting misconfigurations using a tool called `ConfErr`. `ConfErr` is different from our work, `ConfInject`, as we inject misconfigurations that have been reported in bug repositories, while `ConfErr` injects misconfigurations that are derived from error models in human psychology only. We consider `ConfErr` complimentary to `ConfInject`, as both efforts give an indication of how robust a particular software is.

Besides failure characterization and configuration testing, configuration-error detection [31], *Barricade* [35] and diagnosis works like *PeerPressure*[36], *Chronus* [37], *ConfAid* [38] are other areas that have been studied in configuration management research.

X. Conclusion and Future Work

This work presented an empirical study for characterization of configuration problems in Java EE servers, followed by a fault-injection based evaluation. We presented the failure characterization by studying bug reports from the two most popular Java application servers — GlassFish and JBoss. The studied dimensions for each configuration-bug were *problem-type (parameter, compatiblity, missing-component), problem-time (pre-boot-time, boot-time, run-time), problem-manifestation (silent, non-silent)*, and *problem-culprit (developer, user)*. The key findings in our characterization are: (1). More than one-third of problems reported by users are configuration-related implying that configuration-perspective in software developement requires considerable attention. (2). Inter-version problems are majorly parameter related, thus requiring more consistency in configuration-design from version-to-version. (3). Intra-version configuration-problems have an almost equal share for each of *parameter, compatiblity, missing-component* sub-types. (4). Code refactoring or re-implementation results in a increased number of *compatibility* issues.

The second part of this work presented `ConfInject`, a tool that injects misconfigurations based on evidence from bug-repositories. Through the injection process, `ConfInject` automates the configuration-management life-cycle, i.e., start-server, make configuration changes, deploy applications, undeploy applications and stop-server. It subsequently analyzes the reaction of the server to each of the injections by determining the type of manifestation (silent or non-silent) from server logs. JBoss performs better than GlassFish with all of the injections giving a non-silent failure as opposed to only 65% non-silent failures in GlassFish. Based on the results and our analysis, we suggest automated fixing of parameter values.

In the future, we plan to automate the manual process of finding what to inject from bug-repositories. Further, we plan to incorporate run-time injections, i.e., injections while the workload is exercised. We will also see how far the results apply to other kinds of Java software, such as, cluster software, and then software in other programming languages.

REFERENCES

[1] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, ser. USITS'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 1–1.

[2] P. Thibodeau, "Amazon cloud outage was triggered by configuration error," April 2011. [Online]. Available: http://www.computerworld.com/s/article/9216303/Amazon_cloud_outage_was_triggered_by_configuration_error

[3] F. Foo, "Human error triggered nab software corruption," November 2010. [Online]. Available: http://www.theaustralian.com.au/australian-it/human-error-triggered-nab-software-corruption/story-e6frgakx-1225962953523

[4] N. Eddy, "Human error caused google glitch," February 2009. [Online]. Available: http://www.eweek.com/c/a/Enterprise-Applications/Human-Error-Caused-Google-Glitch/

[5] B. Hale, "Why every it practitioner should care about network change and configuration management," February 2012. [Online]. Available: http://web.swcdn.net/creative/pdf/Whitepapers/Why_Every_IT_Practitioner_Should_Care_About_NCCM.pdf

[6] "Glassfish." [Online]. Available: http://glassfish.java.net/

[7] "Jboss." [Online]. Available: http://www.jboss.org/

[8] "Java on openshift." [Online]. Available: https://www.openshift.com/get-started/jboss

[9] L. Keller, P. Upadhyaya, and G. Candea, "Conferr: A tool for assessing resilience to human configuration errors," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, june 2008, pp. 157 –166.

[10] "Java ee jsrs." [Online]. Available: http://jcp.org/en/jsr/platform?listBy=3&listByType=platform

[11] "Oracle jrockit jvm." [Online]. Available: http://www.oracle.com/technetwork/middleware/jrockit/overview/index.html

[12] "Glassfish bug repository." [Online]. Available: http://java.net/jira/browse/GLASSFISH

[13] "Jboss bug repository." [Online]. Available: https://issues.jboss.org/browse/JBAS

[14] "How do i migrate my application from as5 or as6 to as7." [Online]. Available: https://docs.jboss.org/author/display/AS7/How+do+I+migrate+my+application+from+AS5+or+AS6+to+AS7

[15] "Jbas-1115: bad path to included xsd gets built in wsdlfilepublisher." [Online]. Available: https://issues.jboss.org/browse/JBAS-1115

[16] "Geronimo-3921: getcontextroot() returns forward slash rather than empty string for apps deployed to root context." [Online]. Available: https://issues.apache.org/jira/browse/GERONIMO-3921

[17] "Glassfish-6822: Windows: package-appclient must not show forward slash." [Online]. Available: https://java.net/jira/browse/GLASSFISH-6822

[18] "Glassfish-16039: Jms, problem with destination sources having jndi-name with forward-slash." [Online]. Available: https://java.net/jira/browse/GLASSFISH-16039

[19] "Openejb-1709: Tomee webapps (see rest-example) doesn't work under windows (path - problem with backslash '/)." [Online]. Available: https://issues.apache.org/jira/browse/OPENEJB-1709

[20] "Resteasy-656: Context path without trailing slash doesn't work with applicationpath('/')." [Online]. Available: https://issues.jboss.org/browse/RESTEASY-656

[21] "Opensso-544: missing slash in resource names." [Online]. Available: https://java.net/jira/browse/OPENSSO-544

[22] "Sailfin-1556: Slash in sip uri gives error." [Online]. Available: https://java.net/jira/browse/SAILFIN-1556

[23] "Javaserverfaces-1146: Htmlresponsewriter renders unnecessary '/' symbols." [Online]. Available: https://java.net/jira/browse/JAVASERVERFACES-1146

[24] "Glassfish-2778: Treat empty-string virtual-servers attribute in <application-ref> as identical to virtual-servers attribute missing." [Online]. Available: https://java.net/jira/browse/GLASSFISH-2778

[25] "As7-5550: servlet filter mapping empty string not working." [Online]. Available: https://issues.jboss.org/browse/AS7-5550

[26] "Cargo api." [Online]. Available: http://cargo.codehaus.org

[27] "Specjenterprise2010." [Online]. Available: http://www.spec.org/jEnterprise2010

[28] "Glassfish verifier: verify-domain-xml." [Online]. Available: http://docs.oracle.com/cd/E19879-01/820-4337/beadq/index.html

[29] "Suggestimate: Detect duplicate jira issues." [Online]. Available: http://www.plugenta.com/suggestimate/jira/

[30] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy, "An empirical study on configuration errors in commercial and open source systems," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 159–172.

[31] D. Yuan, Y. Xie, R. Panigrahy, J. Yang, C. Verbowski, and A. Kumar, "Context-based online configuration-error detection," in *Proceedings of the USENIX Annual Technical Conference*, 2011, pp. 28–28.

[32] S. Zhang, "Confdiagnoser: an automated configuration error diagnosis tool for java software," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 1438–1440.

[33] Y.-Y. Su, M. Attariyan, and J. Flinn, "Autobash: improving configuration management with operating system causality analysis," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 237–250, Oct. 2007.

[34] J. Li, G. Huang, J. Zou, and H. Mei, "Failure analysis of open source j2ee application servers," in *Quality Software, 2007. QSIC '07. Seventh International Conference on*, 2007, pp. 198–208.

[35] F. Oliveira, A. Tjang, R. Bianchini, R. P. Martin, and T. D. Nguyen, "Barricade: defending systems against operator mistakes," in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 83–96.

[36] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, "Automatic misconfiguration troubleshooting with peerpressure," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 17–17.

[37] A. Whitaker, R. S. Cox, and S. D. Gribble, "Configuration debugging as search: finding the needle in the haystack," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 6–6.

[38] M. Attariyan and J. Flinn, "Automating configuration troubleshooting with dynamic information flow analysis," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–11.