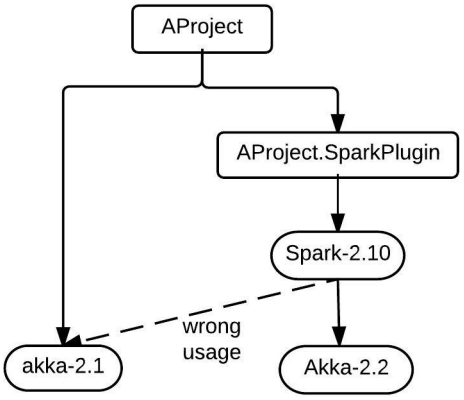
January 21, 2016

1. PROBLEM

To obtain necessary functionality, developers often use third-party libraries. Due to the complexity of current software systems, the dependency issue arises when the system depend on different and incompatible versions of the same library. This issue, called 'dependency hell' [7] may break other dependencies or push the problem to another set of libraries. When developers try to introduce a new library or upgrade existing ones, then other applications on their system might suddenly break as the newly-introduced libraries are not backward-compatible to the existing libraries. We use an example to illustrate how the 'dependency hell' causes a build error, how it is localized, how it is fixed in the next session.

2. EXAMPLE



What is the 'dependency hell'?

We illustrate the building error caused by 'dependency hell' in a Maven project shown in Figure 1. As shown in part A, the parent project uses akka 2.1.1 to initialize `ActorRef` (underline) by creating a new `Props` object. The developer Alice is asked to implement a new sub-module named as SparkPlugin using Spark framework. Without knowing that Spark is dependent on akka-2.2.3, she implements the entire sub-module with Spark-1.0.0, writes the tests, and

(A) Code snippet in parent project using akka 2.1.1

```
/* AProject - pom.xml */
<dependency>
  <groupId>com.typesafe.akka</groupId>
  <artifactId>akka-actor_2.1</artifactId>
  <version>2.1.1</version>
<dependency>

/* AkkaSystemExecutor.java */
ActorSystem system = ActorSystem.create("MySystem");
ActorRef myActor = system.actorOf(new Props(MyUntypedActor.class),"myactor");
```

(B) Code snippet in a sub-module using akka 2.2.3 transitively

```
/* AProject.sparkPlugin - pom.xml */
<parent>
  <artifactId>AProject</artifactId>
  <groupId>..</groupId>...
</parent>

<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.10</artifactId>
  <version>1.0.0</version>
</dependency>
/* SparkUtility.java */
SparkConf sparkConf = new SparkConf();
JavaSparkContext sc = new JavaSparkContext(sparkConf);...
```

(C) Building error when integrating SparkPlugin with parent projects

```
java.lang.NoSuchMethodException: akka.remote.RemoteActorRefProvider.<init>
(java.lang.String, akka.actor.ActorSystem$Settings, akka.event.EventStream,
akka.actor.Scheduler, akka.actor.DynamicAccess)
 at java.lang.Class.getConstructor0(Class.java:3082)...
```

(D) code snippet in spark-core_2.10

```
/* spark-core_2.10, version 1.0.0 pom.xml */
 <properties> ...
    <java.version>1.6</java.version>
    <scala.version>2.10.4</scala.version>
    <akka.group>org.spark-project.akka</akka.group>
    <akka.version>2.2.3-shaded-protob</akka.version>
</properties>
...
<dependency>
  <groupId>$akka.group</groupId>
  <artifactId>akka-remote_$scala.binary.version</artifactId>
</dependency>
//some simplification by tranfering the raw scala code to Java code
ActorRef myActor = system.actorOf(Props.create(MyUntypedActor.class),"myactor");
```

Figure 1: Example from shifu.ml building error

fully tests the single submodule before integrating with the parent project. However, she encounters the building error shown as part (C) (The complete building error is shown at [1]).

How to localize 'dependency hell'?

Starting from the `NoSuchMethodException`, she investigates the `akka.remote.RemoteActorRefProvider.<init>` in akka-2.2.3 as declared in Spark's pom.xml and finds the static `Props.create` method shown in part (D). Alice get confused on the `NoSuchMethodException` and she has to use step-by-step debugging. She finally notices that the maven build system mistakenly uses akka-2.1.1 inherited from parent project, rather than the akka-2.2.3 that is required for Spark library.

How to fix 'dependency hell'?

Alice has several options to fix this bug as below:

1. change akka version in parent class to 2.2+, which is not feasible as the parent project heavily uses akka and it requires tremendous effort for upgrading.

2. exclude all transitive dependencies inherited from parent project with the feature provided by Maven 3.2.2+ [2], aiming to resolve dependency hell [3]. This approach is not feasible as well because SparkPlugin itself relies on the parent projects and Alice has to re-import all dependencies that the submodule uses.

3. include both akka versions and enforce Spark to use akka-2.2.3. This approach is similar to the well-known side-by-side mechanism used in NIX package manager for Linux-based system [6]. This might fix the build error, but Alice is concerned that compiling both versions might increase the building time of the entire system.

4. exclude akka-2.1.1 from the dependency declaration of Spark-1.0.0. Alice decides to use this solution as it won't have ripple effect on the rest of the system.

Using this example, we illustrate that it is not easy to localize dependency hell without knowing the entire dependency graph and it can be error-prone when trying to fix the error.

3. RELATED WORK

When two versions of the same library occur, Maven selects the closest library in the tree of dependencies by default [5] (e.g., if dependencies for A, B, and C are defined as A -> B -> C -> D 2.0 and A -> E -> D 1.0, then D 1.0 will be used when building A because the path from A to D through E is shorter). User is able to manually exclude or force Maven to use a specific version, but the correctness is not guaranteed [2]. Maven Enforcer Plugin checks a set of build rules including a Ban-Transitive-Dependency rule that detects all transitive dependencies conflicts and force users to resolve all conflicts. The Enforcer Plugin only checks the `groupid`, `artifactid`, and `version` declared in the configuration file (TODO: Lisa: though I don't know if we need to check semantics or compatibility of different versions, I leave this limitation for future reference), and it does not help developers fix the conflicts without breaking any other parts of the system. The majority of existing build systems model dependencies identical to Maven. Examples include Gradle, MSBuild, Make, and CloudMake. The features that make them different are unrelated to dependency management, but on syntax of build scripts and parallelization.

Bazel is a new build system developed by Google that advertises parallelization and correctness [4]. Bazel does not allow transitive dependency and only reads dependencies listed in from the root dependency declaration file. This means that if your project (A) depends on another project (B) which list a dependency on project C in its WORKSPACE file, both transitive dependencies from B and C should be added to the WORKSPACE file. This can balloon the file size, but hopefully limits the chances of having one library include C at version 1.0 and another include C at 2.0. But they also assume that users can provide correct configuration and select correct versions of libraries.

Package managers in operating system also encounter similar dependency management problem. To deal with destructive upgrade and multi-distribution support in Linux products, Nix [6] provides a co-existing package management mechanism to store a source package in its own directory, instead of a global location that share across the entire system. However, this might not work in build system like Maven, as with transitive dependencies, the graph of included libraries can quickly grow quite large [5].

4. APPROACH

5. REFERENCES

[1] Build error. In *https://travis-ci.org/lisahua/shifu/builds/29112815*, last visited 1/15/2016.

[2] Maven release note. In *https://maven.apache.org/docs/3.2.1/release-notes.html*, last visited 1/15/2016.

[3] T. O. Brien. Maven-3.2.1 provides a new cure for dependency hell. In *http://discursive.com/2014/03/17/maven-3-2-1-provides-a-new-cure-for-dependency-hell/*, last visited 1/15/2016.

[4] B. Documentation. Transitive dependencies. In *http://bazel.io/docs/external.html*, last visited 1/15/2016.

[5] Maven. Transitive dependencies. In *https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html*, last visited 1/15/2016.

[6] P. Prins, J. Suresh, and E. Dolstra. Nix fixes dependency hell on all linux distributions. In *http://archive09.linux.com/feature/155922*, last visited 1/15/2016.

[7] Wikipedia. Dependency hell. In *https://en.wikipedia.org/wiki/Dependency_hell*, last visited 1/15/2016.