Dynamic class loading

Jan Kolena
(kolena@avast.com)

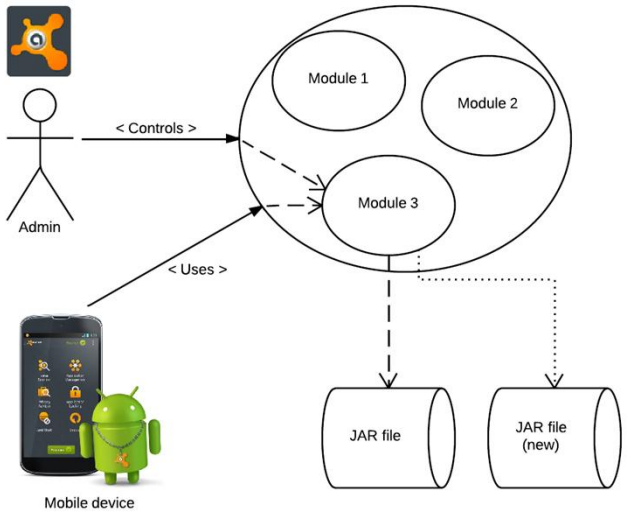
27. 1. 2014

Introduction

- Server application with modules
  - Current load: AVG 2.3k reqs/s, MAX 3.5k reqs/s (total)
- New module – config provider for mobile devices
  - Expected module load: AVG 2k reqs/s, MAX 3k reqs/s (total)
  - Time-various evaluation function (interval days, hours, ..?)
  - The function is generated from pseudocode
    - Auto – tested, but....

Can we trust it?

No.



Dynamic class loading

- Java is flexible (in this way)
- Class file loading is quite friendly, but:
  - Is it satisfying?
  - Is it secure?
  - Is it flexible?

No. 

Distribution of the class

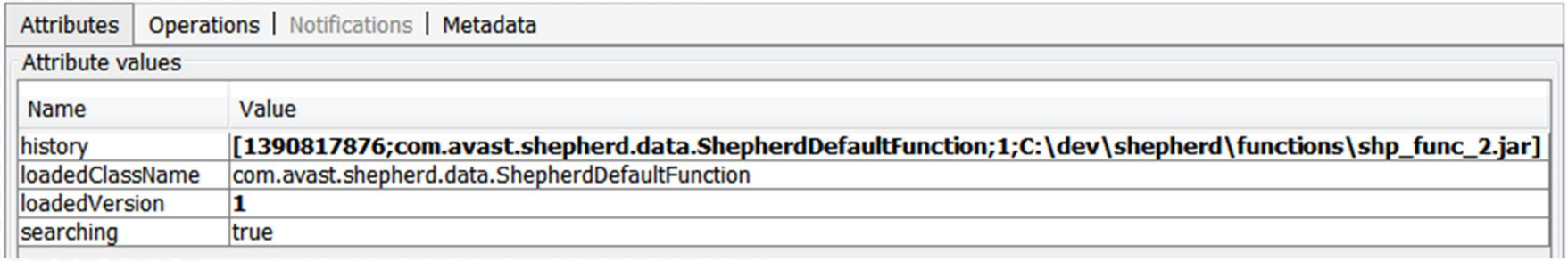
- We already have standard container – the JAR file
- JAR file contains MANIFEST.MF file which:
  - Is not required (by general)
  - Has set of standard attributes (Main-Class, Class-Path, ...)
  - Supports any user-defined attributes (they‘re just skipped, if loader doesn‘t support them)
- Deployment is topic to discuss (FTP, CVS, ...)

Our MANIFEST.MF

```
Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Created-By: Apache Maven
Built-By: kolena
Build-Jdk: 1.7.0_40
Compiled: 1390315821
Main-Class: com.avast.shepherd.data.TestFunction
Implementation-Version: 3
```

JAR loader

- Written in Scala, but „compatible“ with Java users
- Automatic loading of available JARs
  - Root dir, file pattern
  - Own comparator
- Manual loading (from application or by JMX)
- JMX exposed version, name, history etc.



JAR loader usage

```
public class JarLoaderDemo {
    IJarLoader<IDemoFunction> loader = new JarLoader<~>("Demo loader", new File("/root/functions")) {
        @Override
        public void onLoad(IDemoFunction instance, int version, String className, FileSystem fs, Map attributes) {
            System.out.println("New function loaded, class " + className + ", version " + version);
        }
    };

    public JarLoaderDemo() {
        loader.search(10000);
    }

    public static void main(String[] args) {
        new JarLoaderDemo();
    }
}
```

JAR loader interface

avast! be free

```
public interface IJarLoader<T> {

    public void search(int interval, String prefix, String suffix);

    public void init(T defaultInstance, int defaultVersion);

    public void stopSearching();

    public void setComparator(Comparator<File> comparator);

    public boolean load(String name);

    public void acceptOnlyNewer(boolean accept);

    public boolean isSearching();

    public int getLoadedVersion();

    public T getLoadedClass();
}
```

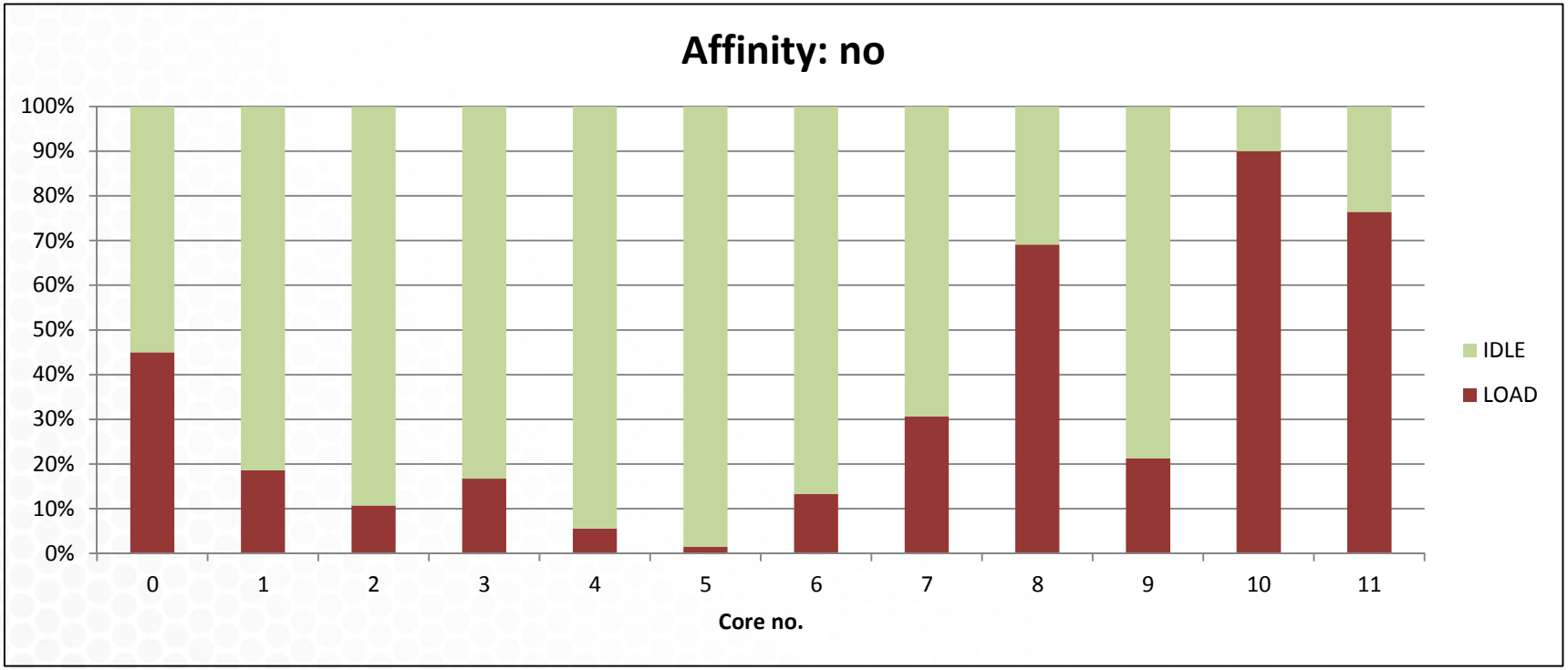
Possible problems

- Race condition
- Harmful code
- Unknown state of application

- CPU consumption
- RAM consumption
- HDD consumption

CPU consumption

- High CPU load can affect whole server
  - e.g. infinite loop (in 4 threads) can take tens of % from each core



  - What about 20 parallel requests?

We have to avoid that!

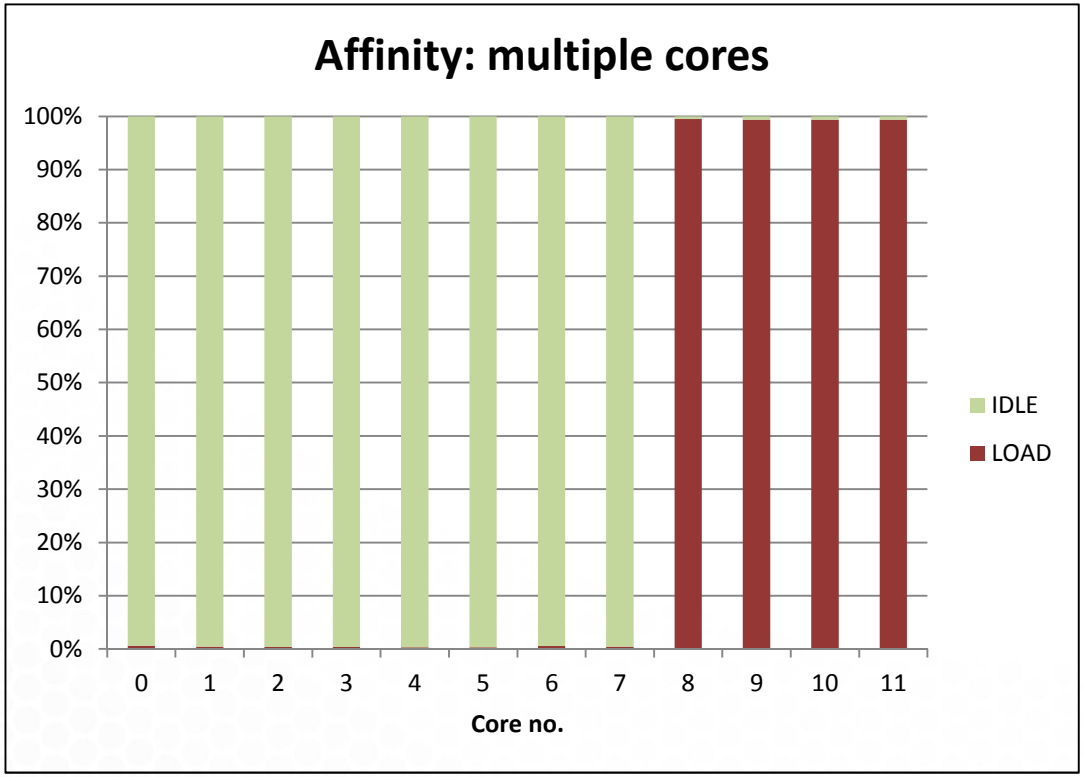
```
public static void main(String[] args) {
    ExecutorService threadPool = new IsolatedThreadPool(CoreStrategy.MULTIPLE, 4);

    for (int i = 0; i < 4; i++) {
        threadPool.execute(() -> { veryHardWorkSimulation(); });
    }
}
```

Each thread has its own core available, but cannot use other cores

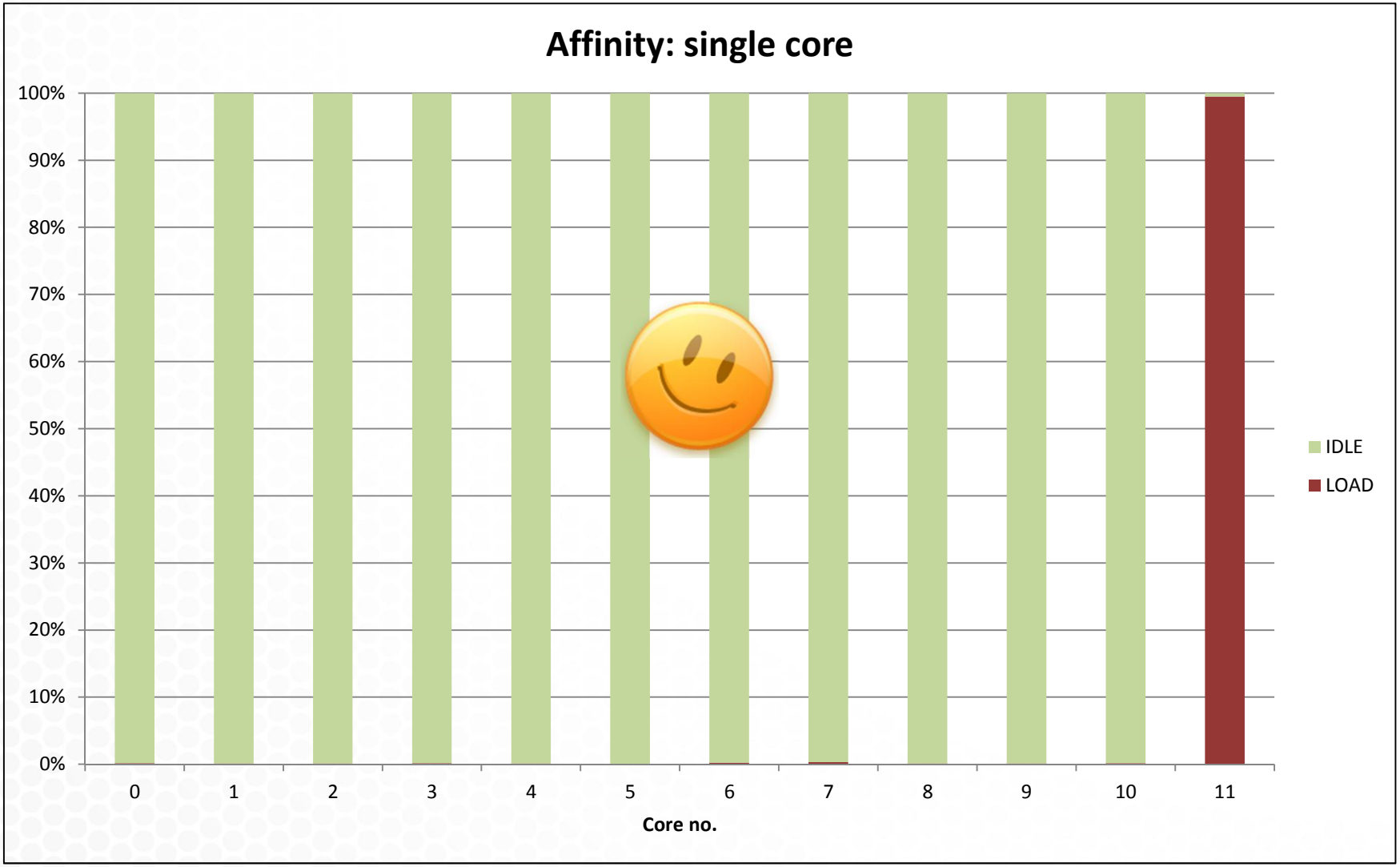
=> thread is isolated

Tested on:

- i7 3740QM @2.7GHz, 16GB RAM, Windows 7 x64
- Xeon E5-2430L @2.00GHz, 48GB RAM, CentOS 6.4 (2.6.32-358.el6.x86_64)



...or we can isolate all threads to a single core:



HDD consumption

- Provide the thread some java.nio.FileSystem (JDK7), which has limited resources (dir with set quota, ...)
  - But we cannot force the thread to use it (not use the default)
- Use security policy
  - Allow access only to directory, which has set quota

RAM consumption

- Even bigger problem than HDD consumption
- Our application has low risc of RAM wasting, but in general..?
- Possible solution: implement own memory manager or JVM (!)

Solution alternatives

- OSGi
  - Powerfull, but... overkill („with a cannon to beat mosquito“)
  - Doesn‘t solve CPU/HDD/RAM consumption
- Standalone JVM connected via Unix sockets/TCP/...
  - **Can solve all introduced problems via JVM run settings and user privilegies**
  - Can/doesn‘t have to use the dynamic loading
  - Increases final latency in case of legal (valid) class loaded
  - Extra application with own lifecycle (disadvantage..?)

Questions?

kolena@avast.com
http://www.linkedin.com/in/jenda

THANKS FOR ATTENTION!

Useful links

- Github.com -> Avast -> jarloader

  https://github.com/avast/jarloader

- JAR loader uses:

  https://github.com/kamranzafar/JCL

- CPU binding uses:

  https://github.com/OpenHFT/Java-Thread-Affinity

- Smileys authors:
  - Everaldo Coelho - http://www.everaldo.com/
  - http://www.2s-space.com/