Exploring Terracotta: JVM-level clustering through Network-Attached Memory

Jonas Bonér

Terracotta, Inc.

jonas@terracotta.org

http://terracotta.org

http://jonasboner.com

- Let’s start with a demo
- – since a picture says more than a thousand words

Agenda

- Overview of Terracotta
- Get started
- Real-world examples
- Q&A

Agenda

- Overview of Terracotta
- Get started
- Real-world examples
- Q&A

What is Terracotta?

Network-Attached Memory

Open Source Scalability and High-Availability for the JVM

What is Terracotta?

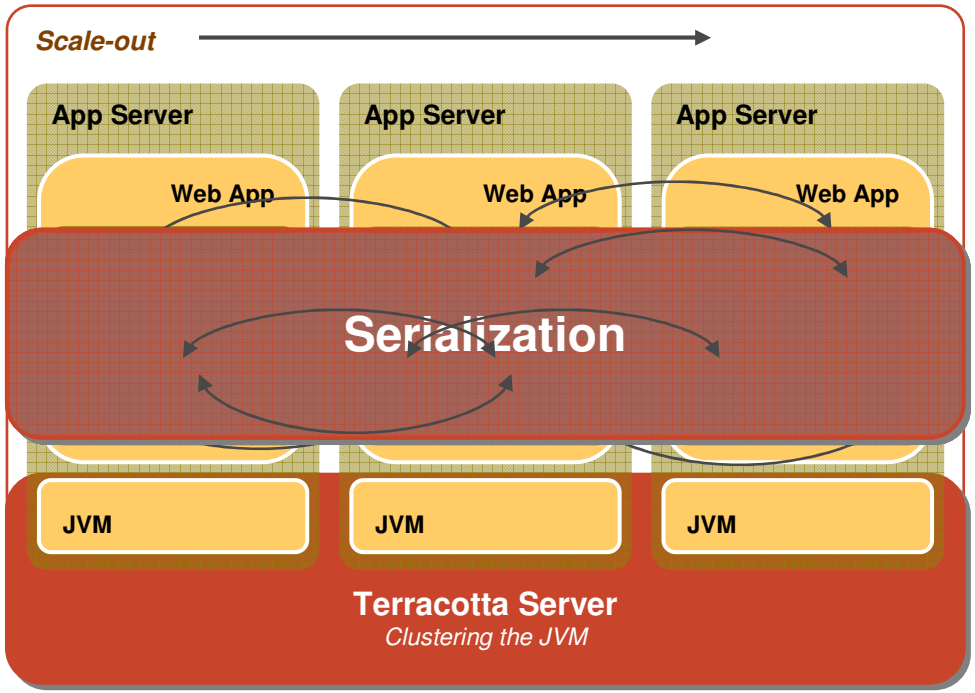
- **Network-Attached Memory**
- **No API**: declarative configuration to selectively share object graphs across the cluster
- **No serialization**: plain POJO clustering
- **Fine-grained replication**: field-level, heap-level replication
- **Cross-JVM coordination**: Java Memory Model maintained across the cluster – e.g. cluster-wide `wait/notify` and `synchronized`
- **Large virtual heaps**: that exceeds the capacity of a single JVM
- Distributed Method Invocations
- Runtime monitoring and control

Use-cases

- **Relieving Database Overload**
  - Distributed Caching
  - Hibernate Clustering
  - HTTP Session Clustering
- **Simplifying Application Architecture and Development**
  - Virtual Heap for Large Datasets
  - Clustering OSS Frameworks (Spring, Struts, Lucene, Wicket, EHCache etc.)
  - Master/Worker – Managing Large Workloads
  - POJO Clustering
  - Messaging, Event-based Systems and Coordination-related Tasks

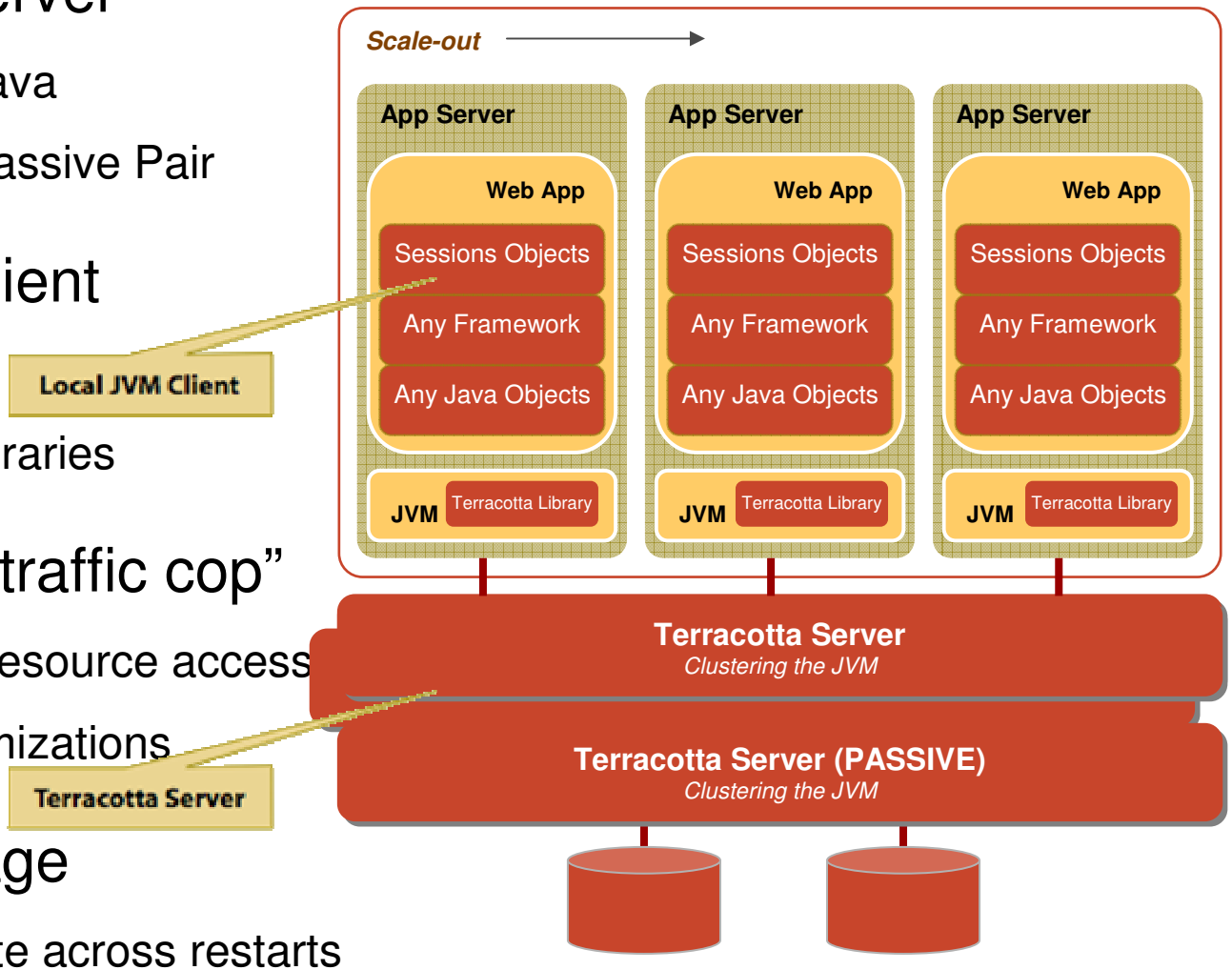
Terracotta approach

- Today's Reality
  - Scale out is complex
  - Requires custom Java code
- Our different approach
  - Cluster the JVM
  - Eliminate need for custom code

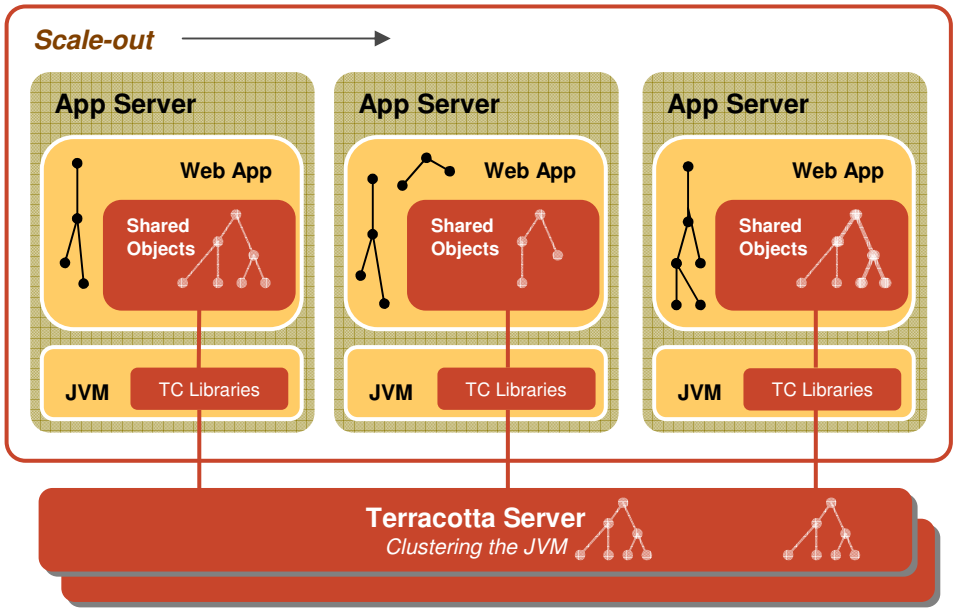


The architecture of Terracotta

- Terracotta Server
  - 100% Pure Java
  - HA Active / Passive Pair
- Local JVM Client
  - Transparent
  - Pure Java Libraries
- Coordinator “traffic cop”
  - Coordinates resource access
  - Runtime optimizations
- Central Storage
  - Maintains state across restarts



Network-Attached Memory



TC Management Console

- Management Console
  - Runtime visibility
  - Data introspection
  - Cluster monitoring

- Heap Level Replication
  - Declarative
  - No Serialization
  - Fine Grained / Field Level
    - `GET_FIELD` – `PUT_FIELD`
  - Only Where Resident
- JVM Coordination
  - Distributed Synchronized Block
  - Distributed `wait()/notify()`
  - Fine Grained Locking `MONITOR_ENTRY` – `MONITOR_EXIT`
- Large Virtual Heaps
  - As large as available disk
  - Dynamic paging

Agenda

- Overview of Terracotta
- Get started
- Real-world examples
- Q&A

Agenda

- Overview of Terracotta
- Get started
- Real-world examples
- Q&A

Hello World

Hello World - tutorial/HelloWorld.java

```
package tutorial;

import java.util.*;

public class HelloWorld {
   private List<String> hellos = new ArrayList<String>();

   public void sayHello() {
      synchronized(hellos) {
         hellos.add("Hello, World " + new Date());
         for (String hello : hellos) {
            System.out.println(hello);
         }
      }
   }

   public static void main(String[] args) {
      new HelloWorld().sayHello();
   }
}
```

Hello World - tc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <application>
    <dso>
      <roots>
        <root><field-name>tutorial.HelloWorld.hellos</field-name></root>
      </roots>
      <locks>
        <autolock>
          <method-expression>* tutorial.HelloWorld*.*(..)</method-
expression>
          <lock-level>write</lock-level>
        </autolock>
      </locks>
      <instrumented-classes>
        <include><class-expression>tutorial..*</class-
expression></include>
      </instrumented-classes>
    </dso>
  </application>
</tc:tc-config>
```

Demo

Enhanced Hello World Using java.util.concurrent

Coordination - HelloWorldConcurrent.java

```
package tutorial;

import java.util.*;
import java.util.concurrent.CyclicBarrier;

public class HelloWorldConcurrent {
   private List<String> hellos = new ArrayList<String>();
   private CyclicBarrier barrier;

   public void sayHello() throws Exception {
      barrier = new CyclicBarrier(2);
      barrier.await();
      synchronized(hellos) {
         hellos.add("Hello, World " + new Date());
         for(String hello : hellos) {
            System.out.println(hello);
         }
      }
   }

   public static void main(String[] args) throws Exception {
      new HelloWorldConcurrent().sayHello();
   }
}
```

Coordination - tc-config-concurrent.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <application>
    <dso>
      <roots>
        <root><field-name>tutorial.HelloWorldConcurrent.hellos</field-name></root>
        <root><field-name>tutorial.HelloWorldConcurrent.barrier</field-
name></root>
      </roots>
      <locks>
        <autolock>
          <method-expression>* tutorial.HelloWorldConcurrent*.*(..)</method-
expression>
          <lock-level>write</lock-level>
        </autolock>
      </locks>
      <instrumented-classes>
        <include><class-expression>tutorial..*</class-expression></include>
      </instrumented-classes>
    </dso>
  </application>
</tc:tc-config>
```

Demo

Agenda

- Overview of Terracotta
- Get started
- Real-world examples
- Q&A

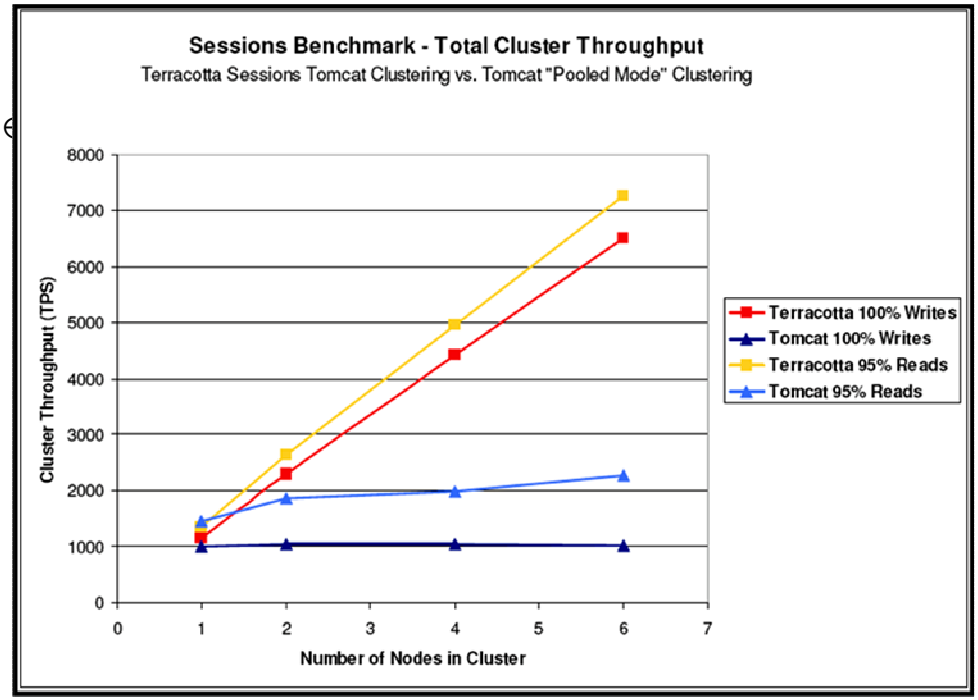
Agenda

- Overview of Terracotta
- Get started
- Real-world examples
- Q&A

HTTP Session Clustering without Java Serialization

HTTP Session Clustering - Benefits

- Terracotta Sessions gives you:
  - Near-Linear Scale
  - No `java.io.Serializable`
  - Large Sessions - MBs
  - Higher Throughput



- Supported Platforms:
  - Jetty,
  - JBoss 4.x,
  - Tomcat 5.0 & 5.5,
  - WebLogic 8.1, WebLogic 9.2,
  - WebSphere CE, Geronimo Alpha, WebSphere 6.1

HTTP Session Clustering - DummyCart.java

```
package demo.cart;

import java.util.*;

public class DummyCart {
    private List items = new ArrayList();

    public List getItems() {
        return Collections.unmodifiableList(items);
    }

    public void addItem(String name) {
        items.add(name);
    }

    public void removeItem(String name) {
        items.remove(name);
    }
}
```

HTTP Session Clustering - carts.jsp

```
<%@ page import="java.util.Iterator" %>
<html>
    <jsp:useBean id="cart" scope="session" class="demo.cart.DummyCart" />
    <%  String submit = request.getParameter("submit");
        String item = request.getParameter("item");
        if (submit != null && item != null) {
            if (submit.equals("add")) {
                cart.addItem(item);
            } else if (submit.equals("remove")) {
                cart.removeItem(item);
            }
        }%>
    <body>
        <p>You have the following items in your cart:</p>
        <ol><%
            Iterator it = cart.getItems().iterator();
            while (it.hasNext()) {
                %><li><% out.print(it.next()); %></li> <%
            } %>
        </ol>

        <form method="get" action="<%=request.getContextPath()%>/cart.jsp">
            <p>Item to add or remove: <input type="text" name="item" /></p>
            <input type="submit" name="submit" value="add" />
            <input type="submit" name="submit" value="remove" />
        </form>
    </body>
</html>
```

HTTP Session Clustering - tc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-
config xmlns:tc="http://www.terracotta.org/config">
  <application>
    <dso>
      <web-applications>
        <web-application>cart</web-application>
      </web-applications>
      <instrumented-classes>
        <include>
          <class-expression>demo.*</class-expression>
        </include>
      </instrumented-classes>
    </dso>
  </application>
</tc:tc-config>
```

Demo

Cluster Spring and other OSS frameworks

Glimpse of supported integrations

- 'Supported' by Terracotta means:
  - Just include a ‘Configuration Module’ (OSGi bundle) in your config to cluster your application (a one liner)
  - Plugs in underneath without any setup
  - Technically, Terracotta supports all integrations as long as it runs on the JVM (f.e. JRuby, PHP)

Example Configuration Module - EHCache clustering

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <clients>
    <modules>
      <module name="clustered-ehcache-1.3" version="1.0.0"/>
    </modules>
  </clients>
  <application>
    <dso>
      <instrumented-classes>
        <include>
          <class-expression>tutorial.*..*</class-expression>
        </include>
      </instrumented-classes>
    </dso>
  </application>
</tc:tc-config>
```

Example - Spring clustering

Terracotta config

```
<spring>
  <application name="tc-jmx">
    <application-contexts>
      <application-context>
        <paths>
          <path>*/applicationContext.xml</path>
        </paths>
        <beans>
          <bean name="clusteredCounter"/>
          <bean name="clusteredHistory"/>
        </beans>
      </application-context>
    </application-contexts>
  </application>
</spring>
```

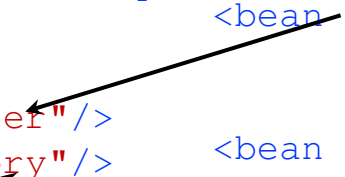
Spring config

```
<bean id="localCounter"
      class="demo.jmx.Counter"/>

<bean id="clusteredCounter"
      class="demo.jmx.Counter"/>

<bean id="localHistory"
      class="demo.jmx.HistoryQueue"/>

<bean id="clusteredHistory"
      class="demo.jmx.HistoryQueue"/>
```



- Terracotta can declaratively cluster Spring beans (Singleton + Session and Custom scoped) with zero code changes
- Can also cluster Spring ApplicationContext events, JMX State and Spring Web Flow

Demo

Cluster Spring Web Flow's
Web Continuations
(conversational/workflow state)

Agenda

- Overview of Terracotta
- Get started
- Real-world examples
- Q&A

Agenda

- Overview of Terracotta
- Get started
- Real-world examples
- Q&A

Wrapping up

Wrapping up

- Terracotta is Network-Attached Memory for the JVM
- Turns Scalability and High-Availability into a **deployment artifact**
- Keep the simplicity of POJO-based development

– get **Scale-Out with Simplicity**

- Makes mission-critical applications **simpler to**:
  - **Write**
  - **Understand**
  - **Test**
  - **Maintain**
- Endless possibilities for clustering and distributed programming – these were just a few
- Be creative, use your imagination and have fun…

Questions?

http://terracotta.org
jonas@terracotta.org

Extra material

Distributed client-side events and data

Distributed client-side - overview

- Terracotta works for all Java apps, not just server-side
- Method invocations can be distributed : DMI

```
<distributed-methods>
  <method-expression>
    void com.MyClass.somethingHappened(String, int)
  </method-expression>
</distributed-methods>
```

- Works out-of-the-box for Swing events and listeners

Distributed client-side - TableDemo.java

```
class TableDemo extends JFrame {
  private DefaultTableModel model;

  private static Object[][] tableData = {
    { " 9:00", "", "", ""}, { "10:00", "", "", ""}, { "11:00", "", "", ""},
    { "12:00", "", "", ""}, { " 1:00", "", "", ""}, { " 2:00", "", "", ""},
    { " 3:00", "", "", ""}, { " 4:00", "", "", ""}, { " 5:00", "", "", ""}
  };

  TableDemo()  {
    super("Table Demo");
    setSize(350, 220);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Object[] header = {"Time", "Room A", "Room B", "Room C"};
    model = new DefaultTableModel(tableData, header);
    JTable schedule = new JTable(model);
    getContentPane().add(new JScrollPane(schedule), java.awt.BorderLayout.C
ENTER);
  }

  public static void main(String[] args) {
    new TableDemo().setVisible(true);
  }
}
```

Distributed client-side - tc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <application>
    <dso>
      <instrumented-classes>
        <include>
          <class-expression>demo.*..*</class-expression>
        </include>
      </instrumented-classes>
      <roots>
        <root>
          <field-name>demo.TableDemo.model</field-name>
        </root>
      </roots>
    </dso>
  </application>
</tc:tc-config>
```

Demo

Fine-grained Distributed Caches

Distributed Caches - Benefits

- Simple : pick the cache and just declare the module
  - EHCache, JBoss TreeCache, OSCache
  - Java Collection (e.g. java.util.Hashmap, java.util.HashSet, java.util.TreeMap, ...)
- Fast:
  - Field Level Delta changes
  - Replicate only where needed
  - Appliance-like design that can optimize itself
- Big:
  - Virtual heap that exceeds limitations of single JVM
  - Lazy-loading with on-demand faulting and flushing of data
  - Coherent

Distributed EHCache - EHCache.java

```
package tutorial;
import net.sf.ehcache.*;

public class EHCache {
  private CacheManager cacheManager = CacheManager.create("ehcache.xml");
  private Cache cache;

  public EHCache() {
    this.cache = cacheManager.getCache("TestCache");
    if (null == this.cache) {
      Cache cache = new Cache("TestCache", 1000, false, false, 120, 120);
      cacheManager.addCache(cache);
      this.cache = cache;
    }
  }

  public void doCache() {
    cache.put(new Element(new java.util.Date(), System.currentTimeMillis()));
    for (Object key : cache.getKeys()) {
      System.out.println(cache.get(key));
    }
  }

  public static void main(String[] args) {
    new EHCache().doCache();
  }
}
```

Distributed Ehcache - tc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <clients>
    <modules>
      <module name="clustered-ehcache-1.3" version="1.0.0"/>
    </modules>
  </clients>
  <application>
    <dso>
      <instrumented-classes>
        <include>
          <class-expression>tutorial.*..*</class-expression>
        </include>
      </instrumented-classes>
    </dso>
  </application>
</tc:tc-config>
```

Demo