POJO Scalability and Large Workloads with Terracotta

Jonas Bonér

Terracotta, Inc.

jonas@terracottatech.com

http://jonasboner.com

TERRACOTTA

Who is Jonas?

- Hacker, OSS evangelist, Agile Practitioner
- Founder of AspectWerkz
- Committer to Eclipse AspectJ
- Committer to Terracotta
- Ski and Jazz fanatic
- Currently learning:
    - Haskell, Erlang
    - How to become a better dad

Goal of this session

- Learn how JVM-level clustering and Terracotta works at a high level
- Learn how use it to scale-out POJO-based applications using Master/Worker and Locality of Reference

Agenda

1. **Grids - What's Behind the Buzz?**
2. **Master/Worker Pattern**
3. **JVM-level Clustering with Terracotta**
4. **Case-study – Distributed Web Spider:**
    1. **Master/Worker Container (POJO-based, single JVM)**
    2. **Web Spider Implementation**
    3. **Cluster It Using JVM-level Clustering**
    4. **Run It as a Grid**
5. **Real-World Challenges**
6. **Questions**

What is a Grid?

Here is one definition:

“A *Grid* is a set of servers **that together creates a mainframe class processing service where** data and operations can move seamlessly **across the grid in order to** optimize the performance and scalability **of the computing tasks submitted to the grid.”**

How do Grids scale?

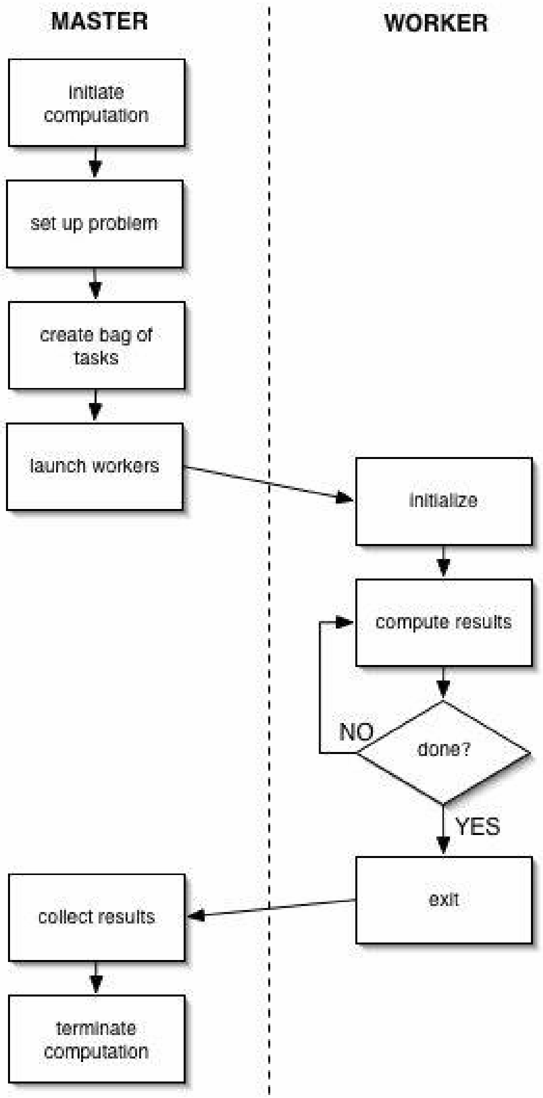
- **Make use of Locality of Reference**
  - **Data local to a specific node stays there**
  - **Move operations around instead of data**
  - **Move the application to the data**
- **Work partitioning**
  - **Ultimate: Work is “Embarrassingly Parallel” - no shared state**
  - **Acceptable: Partition the work into logical groups working on the same data set**

How do Grids handle failure?

- **Highly available using data duplication**
- **Grids are build to expect failure**
  - **In contrast to "traditional" distributed computing in which every component has to expect the worst and protect itself accordingly**
- **Automatically re-executes pending and failed work**

Grids: Master/Worker in a Box?

Master/Worker pattern



- **1 Master**
- **1-N Workers**
- **1 Shared Memory Space**
- **Common applications**
  - **Financial Risk Analysis and other Simulations**
  - **Searching / aggregation on large datasets**
  - **Sales Order pipeline processing**

How can we implement Master/Worker in Java?

- **Concurrency primitives allows you to write your own implementation**
  - **`wait/notify – synchronized` blocks etc.**
  - **Might be tricky to implement correctly and to achieve good performance**

How can we implement Master/Worker in Java?

- `java.util.concurrent.ExecutorService`

  - **Highly tuned, high-level abstractions**
  - **Direct support for *Master/Worker* pattern**

  Problems:

  - **Does not separate *Master* from *Worker***
  - **Provides no information about *Work* status**

How can we implement Master/Worker in Java?

- **CommonJ WorkManager**
  - **IBM and BEA specification that allows threading in JEE**

Advantages:

- **Still simple POJO based**
- **Can wrap Java 5 concurrency abstractions**
- **Gives us the right abstraction level**
- **Allows us to add a layer of reliability**

Review the Goal

- **What we want to do:**
  1. **Implement a thread-based Master/Worker container**
  2. **Distribute out Workers (and Masters) onto multiple JVMs**
  3. **Ensure application performance by minimizing data movement payload across worker contexts**
- ***CommonJ WorkManager* seems to be up for the task, but…**
- **How can we do this?**
- **Can we use clustering?**

Yes, clustering is a solution - but we want: Simplicity **and** Scale-out

- **Simplicity**
  - **No usage of proprietary APIs**
  - **Preservation of Object Identity - no serialization, works with POJOs**
  - **Preservation of the semantics of the JLS and JMM**
- **Scale-out**
  - **Fine-grained and lazy replication**
  - **Runtime lock optimization for clustering**
  - **Runtime caching for data access**

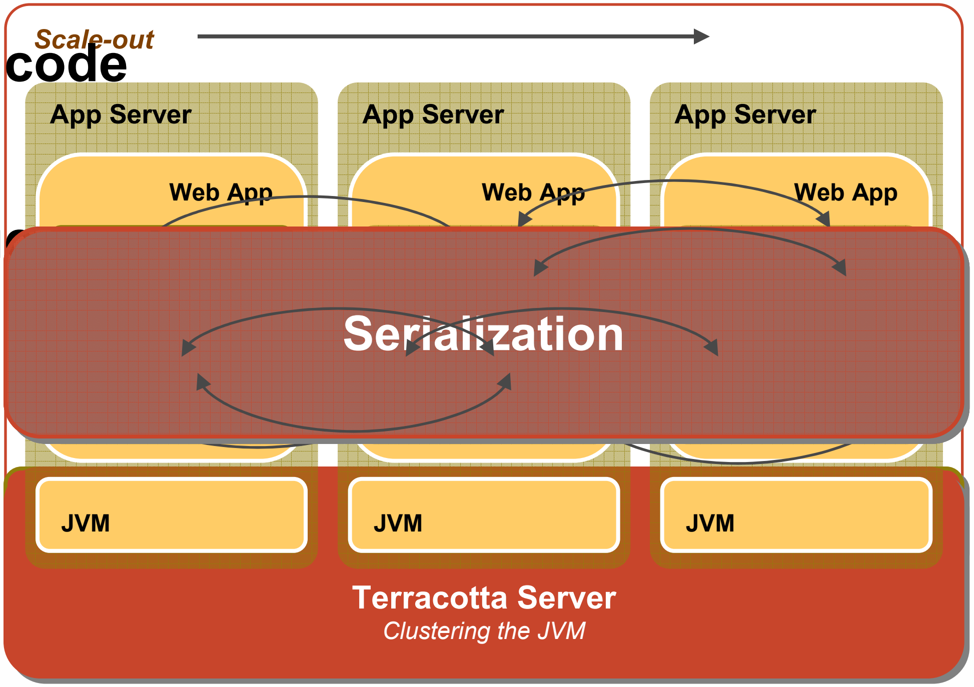
The ideal solution: JVM-level clustering

Enter Terracotta

- Delivers clustering as a runtime infrastructure service - a deployment artifact
- Clusters the JVM
- Open Source under Mozilla-based license

Terracotta approach

- Today's Reality
  - Scale out is complex
  - Requires custom Java code
- Our approach is fundamentally different
  - Cluster the JVM
  - Eliminate need for custom code
- Development Benefits
  - Leverage existing infrastru
  - Substantially less code
  - Focus on business logic
  - Consistent solution
- Operational Benefits
  - Scale independently
  - Consistent and manageable
  - Provides increased visibility

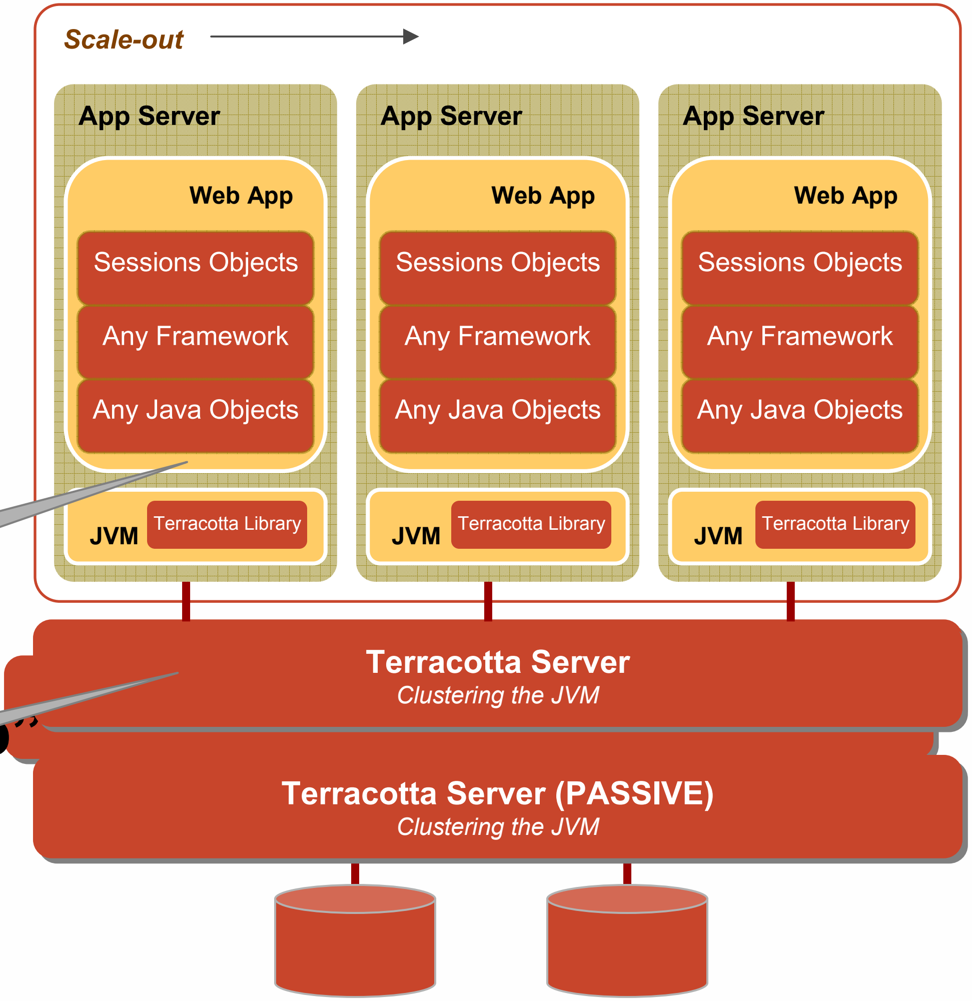


Terracotta Use Cases

HTTP Session Clustering	eTail - HA for Shopping Cart Telco - HA for User Sessions SAAS - Online Testing Services	BT NTT Data SEI New ways. New answers.
Distributed Caching	Mobile - Mobile Search Content Media - Content Aggregation Publication - Content Caching Financial Services - Matching Engine Financial Services - Trading Application Logistics - Reporting Applications Etail - Catalog	comcast 1UP.com mcn pasenger ZIFF DAVIS MEDIA
Clustering POJO’s	Healthcare - Availability of Patient Information Online Gaming - Customer Account Balance Publishing - Reference Data Manufacturing - Dealership Inventory	PEARSON xuqa.com beta
Clustering Spring	eTail - Ticketing and Seating Availability Construction - Financial Reporting	premières loges logiciels de billetteries SHAWMUT
Collaboration / Coordination / Eventing	eTail - Order Processing Financial Service - Order Processing Financial Services / Telco - Data Grid Online Gaming - Game Table Coordination	PartyPoker.com The World's Largest Poker Room

Terracotta architecture

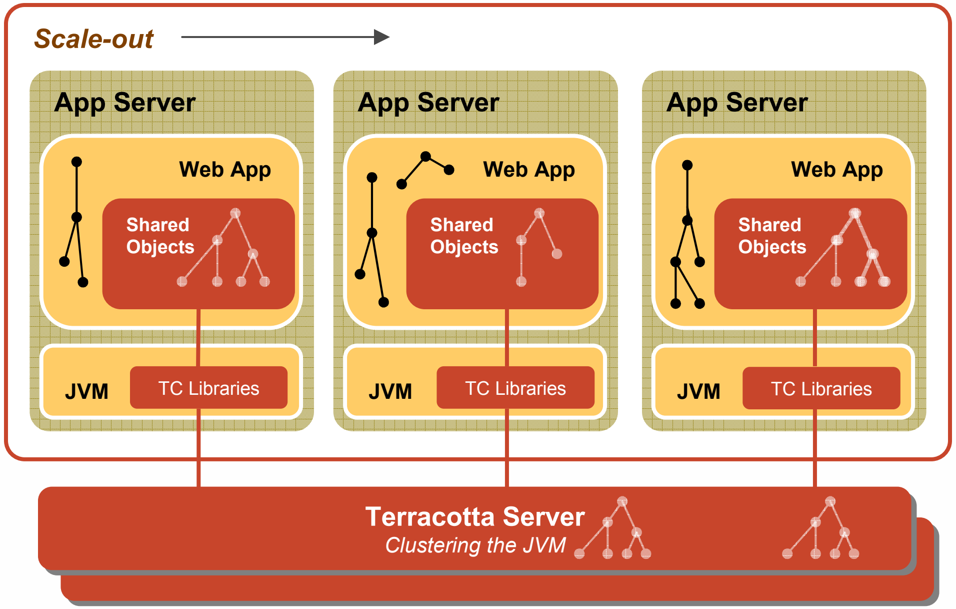
- **Terracotta Server**
  - **100% Pure Java**
  - **HA Active / Passive Pair**
- **Local JVM Client**
  - **Transparent**
  - **Pure Java Libraries**
- **Central Storage**
  - **Maintain**
    restarts
- **Coordinator “traffic cop”**
  - **Coordina**
    access
  - **Runtime optimizations**



Local JVM Client

Terracotta Server

Terracotta Features



- 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

Terracotta Usability Features

- Configuration
  - Declarative
  - Configuration Modules
- Developer / Tuning Tools
  - Eclipse Integration
  - Configurator
  - Error Reporting
  - Application Analyzer (upcoming)
  - Deadlock Detection (upcoming)
- Operational Tools
  - JMX Support
  - Cluster Membership
- Administration Console
  - Cache Hits
  - Transactions
  - Shared Objects / Object Graphs
  - Shared Classes

jetty://
RIFE
Lucene
Spring
Glassbox
JUST TELLS YOU WHAT BROKE™
Quartz
Enterprise Job Scheduler
maven
EHCACHE
RoMA
WICKET
APACHE GERONIMO
eclipse
OPENSYMPHONY
QUALITY COMPONENTS
HIBERNATE
Struts
OSGi Alliance
iBATIS
red5
IBM

The power of JVM-level clustering

- **Clustering the JVM underneath *CommonJ WorkManager* delivers POJO-based Grid:**
  - Simplicity:
    - **POJOs - Standard JDK 1.5 code**
  - Performance:
    - **Locality of Reference + fine-grained replication**
  - Scale-Out:
    - **Ability to scale Masters and Workers independently**
  - High-Availability:
    - **Data resides on the “network” - fail-over to any other node**

Demo: Master/Worker

Case study

1. **Implement a Master/Worker “container”**
2. **Implement a Web Crawler that uses our “container”**
3. **Cluster it with Terracotta**
4. **Look into how we can tackle some real-world challenges**

CommonJ WorkManager specification 1

```
public interface Work extends Runnable {
}

public interface WorkItem {
    Work getResult();
    int getStatus();
}
```

CommonJ WorkManager specification 2

```
public interface WorkManager {
    WorkItem schedule(Work work);
    WorkItem schedule(Work work,
  WorkListener listener);
    boolean waitForAll(Collection workItems,
  long timeout);
    Collection waitForAny(Collection
  workItems, long timeout);
}
```

CommonJ WorkManager specification 3

```
public interface WorkListener {
    void workAccepted(WorkEvent we);
    void workRejected(WorkEvent we);
    void workStarted(WorkEvent we);
    void workCompleted(WorkEvent we);
}
```

CommonJ WorkManager specification 4

```
public interface WorkEvent {
    int WORK_ACCEPTED  = 1;
    int WORK_REJECTED  = 2;
    int WORK_STARTED   = 3;
    int WORK_COMPLETED = 4;
    public int getType();
    public WorkItem getWorkItem();
    public WorkException getException();
}
```

1. Let’s look at the code for Master/Worker

2. Implementing a Web Spider

- **What is a Web Spider?**
  1. **Grabs the page from a URL**
  2. **Does something with it – for example indexes it using *Lucene***
  3. **Parses it and find all URLs from this page**
  4. **Grabs these pages**
  5. **Parses them and…so on…you get the idea**
- **How to slice the problem?**
  1. **Create new *Work* for a URL to a page to parse**
  2. **Pass it to the *WorkManager***
  3. **When executed, the *Work* grabs the page, parses it and gathers all its URLs**
  4. **For each new URL: GOTO 1.**
- **We are using the Master/Worker “container” to parallelize the work**

3. Cluster with Terracotta

- **Do not change the application**
- **Declaratively select which objects should be shared across the grid**
  - **E.g. which part(s) of the Java heap that should be always up-to-date and visible to all parts of the application that needs it – in the whole grid**

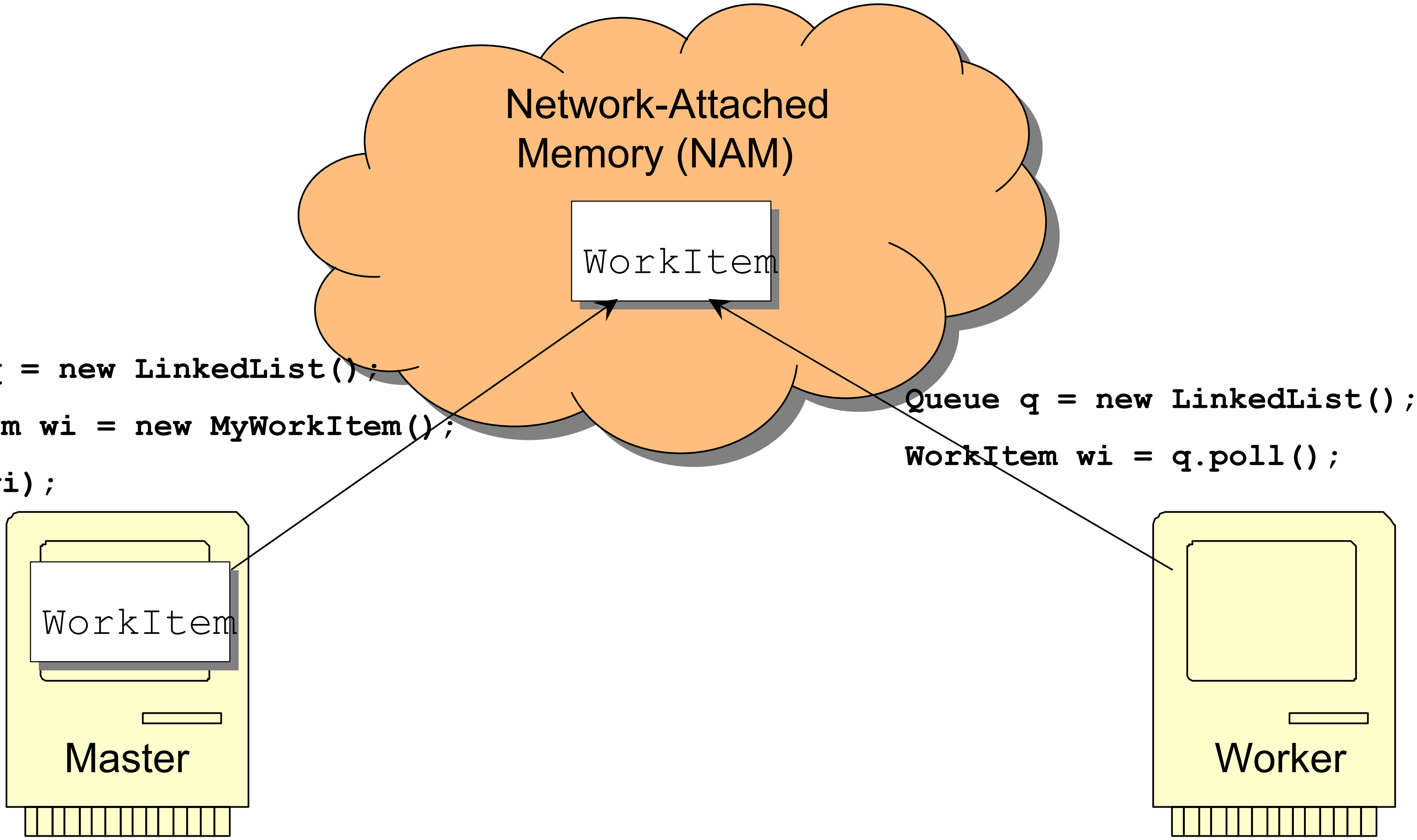
Terracotta configuration

define *roots*

define *includes*

```
<roots>
  <root>
    <field-name>
     org.tc.workmanager.SingleWorkQueue.m_workQueue
    </field-name>
  </root>
</roots>
<instrumented-classes>
  <include>
    <class-expression>org.tc.workmanager..*</class-expression>
  </include>
  <include>
    <class-expression>org.tc.spider..*</class-expression>
  </include>
</instrumented-classes>
```

Master and Worker are operating on **the exact same** but **still local** `WorkItem` instance



```
Queue q = new LinkedList();
WorkItem wi = new MyWorkItem();
q.put(wi);
```

```
Queue q = new LinkedList();
WorkItem wi = q.poll();
```

4. Challenges

- Routing?
- How to handle work failure?
- Ordering matters?
- Worker failure?
- Very high volumes of data?

Routing

- **Keep state in the *Work* – no state in *Worker***
- **Route *Work* that are working on the same data to the same node**
- **Work can repost itself or new work onto the *Queue* and is guaranteed to be routed to the same node**

```
 public class RoutableWorkItem<ID> extends
      DefaultWorkItem implements
Routable<ID> {
    protected ID m_routingID;
    ...
 }
```

Routing

```
public interface Router<ID> {
  RoutableWorkItem<ID> route(Work work);
  RoutableWorkItem<ID> route(Work work, WorkListener listener);
  RoutableWorkItem<ID> route(RoutableWorkItem<ID> workItem);
}
```

- **Can use different load-balancing algorithms**
  - **Round-robin**
  - **Work load sensitive balancing (*Router* looks at *Queue* depth)**
  - **Data affinity - “Sticky routing”**
  - **Your own…**

Retry

- Retry on failure
- Event-based failure reporting
- Use the `WorkListener`

```
 public void WorkListener#workRejected(WorkEvent
we);

 public void workRejected(WorkEvent we) {
   Expection cause = we.getException();
   WorkItem wi = we.getWorkItem();
   Work work = wi.getResult();
   ... // reroute the work onto queue X
 }
```

Ordering matters?

1. **Use a `PriorityBlockingQueue<T>` (instead of a `LinkedBlockingQueue<T>`)**
2. **Let your `Work` implement `Comparable`**
3. **Create a custom `Comparator<T>`:**

```
Comparator c = new Comparator<RoutableWorkItem<ID>>() {
   public int compare(
         RoutableWorkItem<ID> workItem1,
         RoutableWorkItem<ID> workItem2) {
     Comparable work1 =
        (Comparable)workItem1.getResult();
     Comparable work2 =
        (Comparable)workItem2.getResult();
     return work1.compareTo(work2);
   }};
```

4. **Pass it into the constructor of the `PriorityBlockingQueue<T>`**

Worker failure detection: approaches

- **Heartbeat mechanism**
- **Work timestamp – Master checks for timeout**
- **Worker holds an “is-alive-lock” that Master tries to take**
- **Notification from Terracotta Server (since 2.3)**
- **If detected: reroute all non-completed work**

Very high volumes of data?

- Problem: **Bottlenecks on the single *Queue***
  - **High contention + Bad Locality of Reference**

- Solution:

1. **Create a *Channel* abstraction**
   - **Has 2 queues - pending and result**
2. **Each *Worker* has its own *Channel(s)***
3. **Load-balancing in the *Master(s)***

→ **Maximizes Locality of Reference**

→ **Minimizes contention**

Very high volumes of data – Result 1

Single Queue Implementation

~ 100 TPS (regardless of # nodes)

Channel Implementation

1 Node : 600 TPS

2 Nodes : 750 TPS

3 Nodes : 1000 TPS

Very high volumes of data - Batching

- Better, but still not great throughput
- Solution:
  - **Use Batching**
  - **Create a configurable *BatchingChannel***

Very high volumes of data – Result 2

Single Queue Implementation

~ 100 TPS (regardless of # nodes)

Channel Implementation

1 Node : 600 TPS

2 Nodes : 750 TPS

3 Nodes : 1000 TPS

Channel Implementation with Batching

1 Node : 1000 TPS

2 Nodes : 1750 TPS

3 Nodes : 2500 TPS

Wrap up: developer benefits

- Work with plain POJOs
- Event-driven development
  - Does not require explicit threading and guarding
- Test on a single JVM, deploy on multiple JVMs
- White box implementation
  - Freedom to design Master, Worker, routing algorithms, fail-over schemes etc. the way you need

Learn more

- **Checkout the source:**
  - http://svn.terracotta.org/svn/forge/projects/labs/opendatagrid (simple)
  - http://svn.terracotta.org/svn/forge/projects/labs/workmanager (performant)
- **Download Open Terracotta today:**
  - http://terracotta.org
- **Articles:**
  - http://jonasboner.com/2007/01/29/how-to-build-a-pojo-based-data-grid-using-open-terracotta/
  - http://www.theserverside.com/tt/articles/article.tss?l=DistCompute
- **Documentation and blogs:**
  - http://terracotta.org
  - http://blog.terracottatech.com/
  - http://jonasboner.com

Questions?

Thanks

http://terracotta.org