1. Session Migration & Replication - WADI vs Terracotta

Introduction

One of WADI's strengths is its distributed session lookup engine, which provides the necessary infrastructure to track session locations in a constant cost whatever the number of nodes in a cluster. Thanks to this design, the cost of a session insertion, migration or invalidation is constant, once again, whatever the number of cluster members.

A second strength is its replication engine, which only replicates sessions to a configurable number of nodes. As a consequence, replication overhead is solely a function of the number of back-ups to be maintained. In other words, replication remains efficient whatever the number of cluster members.

The main objective of the scalability and performance tests presented by this report is to validate and demonstrate the effectiveness of WADI's distributed session engine and replication engine through specific experimentations.

A subsidiary goal is to measure some of WADI's performance characteristics and to compare them with the characteristics of another clustering solution, Terracotta, which also scales up to multiple nodes without noticeable performance impacts.

This report includes average response times and CPU usage comparisons between WADI and Terracotta, which are to be read and interpreted in the described contexts. More explicitly, the fact that WADI's average response times are smaller than Terracotta's ones in the considered usage scenarios certainly does not mean that WADI is generally faster than Terracotta.

Tests Description

Session Migration Test

- Replication is disabled for WADI <= Is Equivalent to => One Terracotta Server is used
- The WADI demonstration Web-application is used
- This web-application is deployed to an increasing number of Jetty instances clustered by WADI or Terracotta
- A load-balancer, HAProxy (http://haproxy.1wt.eu/), distributes inbound requests to these downstream Jetty instances based on a round-robin policy. The load-balancer is intentionally not configured for HTTP session stickiness in order to trigger a session migration each time that a request is sent to the cluster through the load-balancer.
- A client GETs the resource "http://localhost:<port>/wadi-webapp/session.jsp?limit=9" 5,000 times. The first GET triggers the creation of a session. In this session are stored two objects:
    - a Counter, simple wrapper around the int primitive, which is used to count the number of times session.jsp has been hit; and
    - a LinkedList, which is used to store an history of the nodes having served the last 9 GETs.
      Based on these two objects, the proper test execution can be confirmed: firstly, the number of GETs received by the cluster is checked based on the Counter value; and the even distribution of the GETs to the downstream Jetty instances is verified based on the history of the nodes having served the last 9 GETs.
- The response time of these GETS, generated by Grinder (http://grinder.sourceforge.net/), are captured for later analysis
- The last GET response is written to a file and manually checked to confirm the overall success of the test execution

- The 5,000 GETs are executed twice: a first time to warm the targeted Jetty instances and a second time during which response times and CPU statistics are measured
- Grinder, Jetty instances and Terracotta servers are all running on the same physical box, a MacBook Pro having a 2.16GHz dual core processor and 2GB of memory
- CPU statistics are captured every 5 seconds via iostat. Memory and io resource usage could have also been captured via vm_stat or netstat respectively. However, their results would not have been pertinent. Indeed, the scenario is not memory intensive; furthermore, communications happen over the loopback interface which is sufficiently fast to become a bottleneck after the exhaustion of CPU resources.

Session Migration and Replication Test

- Same as "Session Migration Test" except for the replication part
- Replication is enabled and only one replica is maintained by WADI <= Is Equivalent to => Two Terracotta servers in an active-passive over network configuration are defined

Session Replication Test

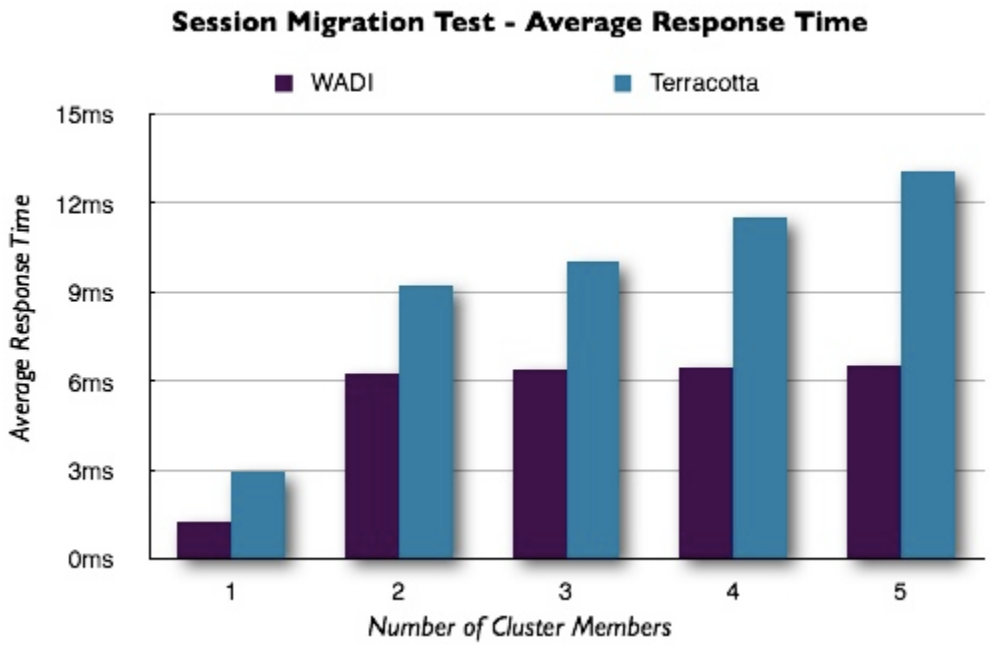
- Same as "Session Migration and Replication Test" except for the load-balancing part
- The load-balancer is configured for HTTP session stickiness.

Results

Session Migration Test

Average Response Time

Graph

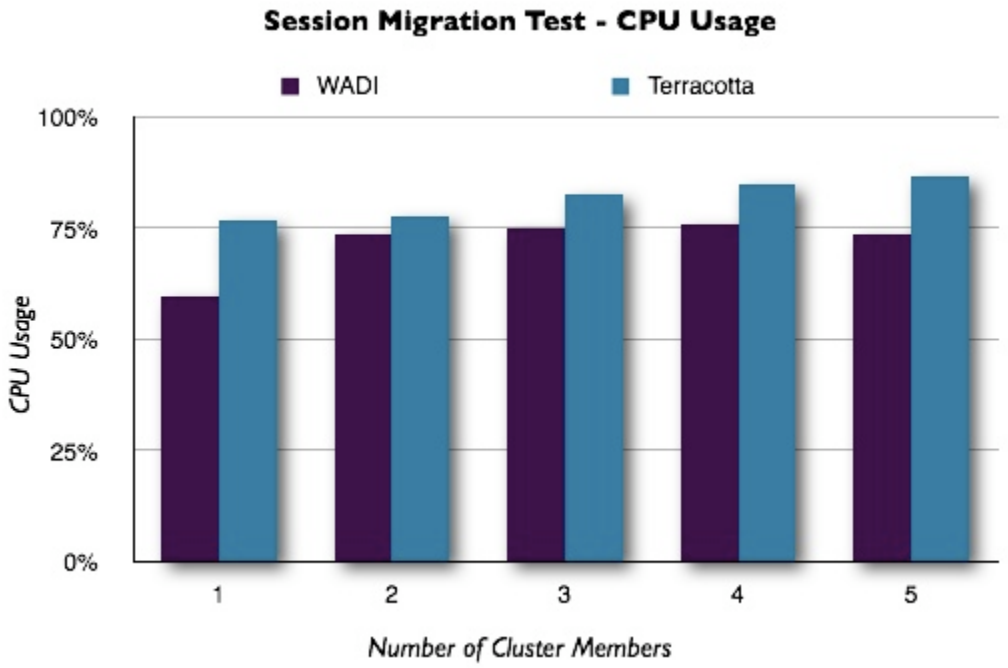


Raw Data

1. of nodes	Average Response Time - WADI	Average Response Time - Terracotta
1	1.26	2.95
2	6.29	9.23
3	6.39	10.10
4	6.52	11.57
5	6.53	13.12

CPU Usage

Graph



Raw Data

1. of nodes	Average CPU Usage - WADI	Average CPU Usage - Terracotta
1	60	77
2	74	78
3	75	83
4	76	85

5	74	87

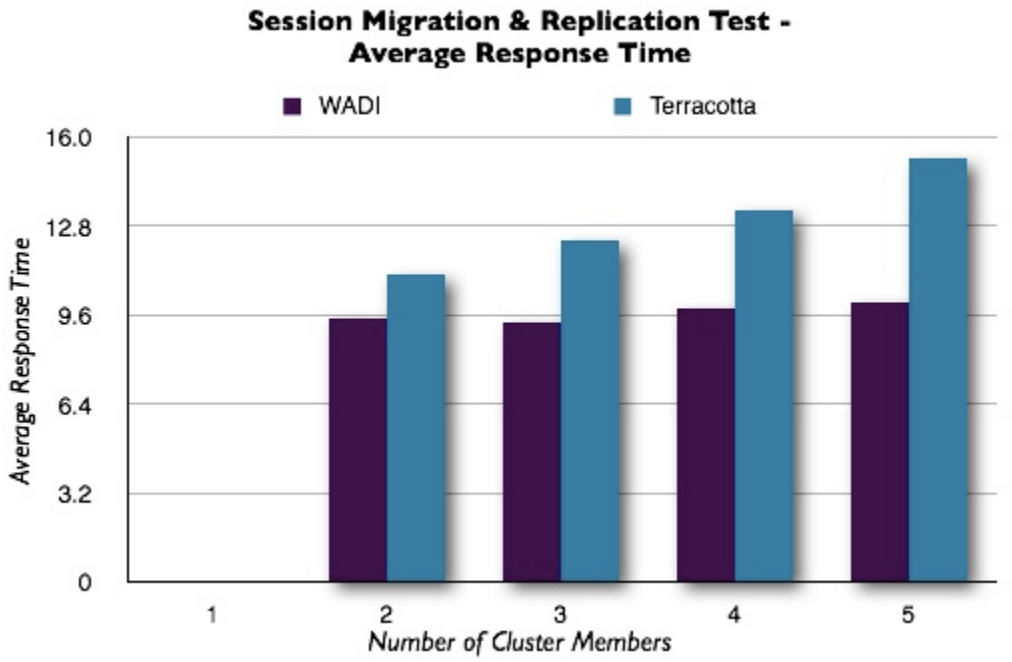
Observations

- As expected, the cost of a session migration, i.e. the cost of moving the state of a session form one node to another node, remains constant while the number of nodes increases for WADI.
- GETs served by Jetty instances clustered by WADI are between 32% to 50% faster than Terracotta - for the considered scenario.
- WADI is less CPU intensive than Terracotta - for the considered scenario.

Session Migration and Replication Test

Average Response Time

Graph



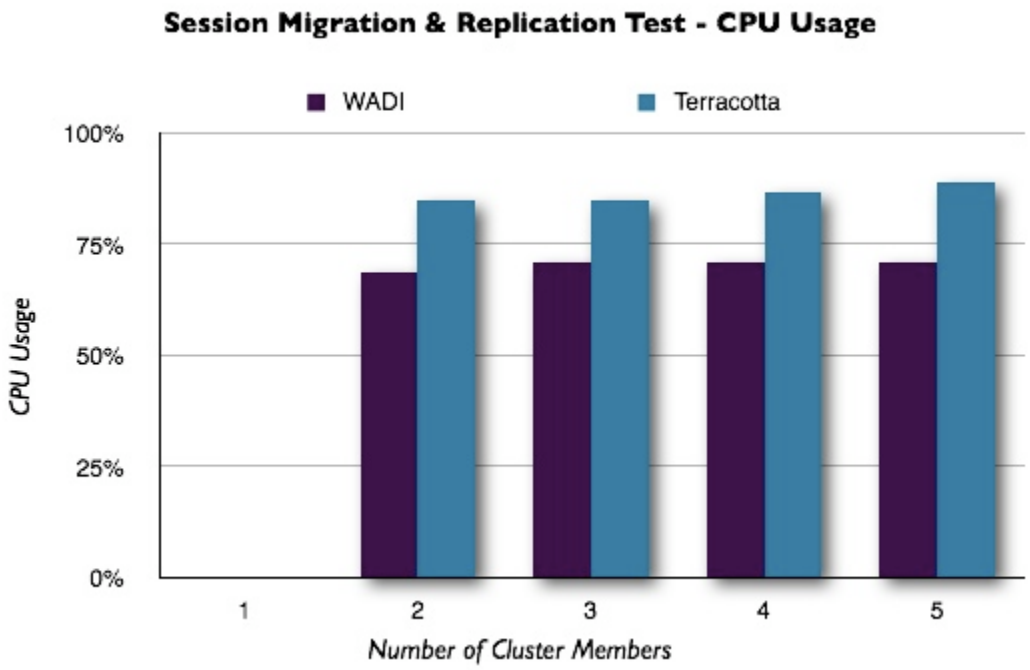
Raw Data

1. of nodes	**Average Response Time - WADI**	**Average Response Time - Terracotta**
1	N/A	N/A
2	9.49	11.08
3	9.40	12.32
4	9.85	13.42

5	10.07	15.28

CPU Usage

Graph



Raw Data

1. of nodes	Average CPU Usage - WADI	Average CPU Usage - Terracotta
1	N/A	N/A
2	69	85
3	71	85
4	71	87
5	71	89

Observations

- One again, the cost of keeping one session replica, i.e. the cost of keeping a copy of a session on an another node than the one currently owning the session, remains constant while the number of nodes in the cluster increases for WADI.
- GETs served by Jetty instances clustered by WADI are between 14% to 34% faster than Terracotta - for the considered scenario.
- WADI is less CPU intensive than Terracotta - for the considered scenario.

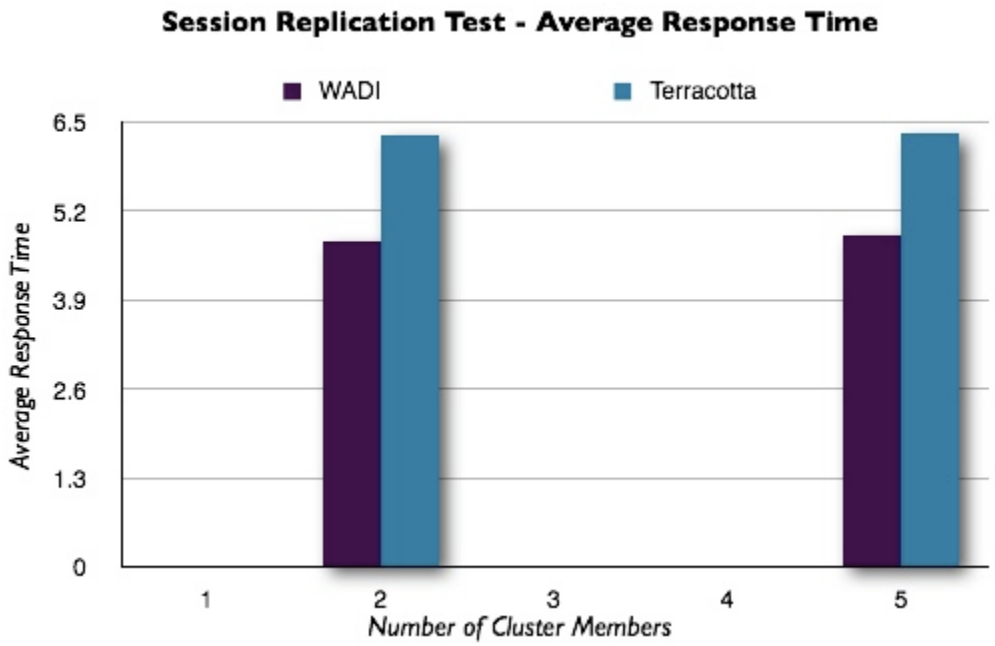
Note on WADI Replication Strategy

- Out-of-the-box, WADI provides a replication strategy allowing the configuration of the number of replicas to be maintained for a given session.
- Replicas are automatically re-organized when new nodes join or leave the cluster. For example, when the number of replicas is set to two and a node joins a cluster of two nodes, new replicas are automatically added to the joining node to guarantee the existence of 2 replicas per session.
- Users requiring specific replication strategies can easily plug-in their own strategy. A specific replication strategy could be: session replicas are to be hosted by nodes running on distinct hardware than the node owning the session.

Session Replication Test

Average Response Time

Graph



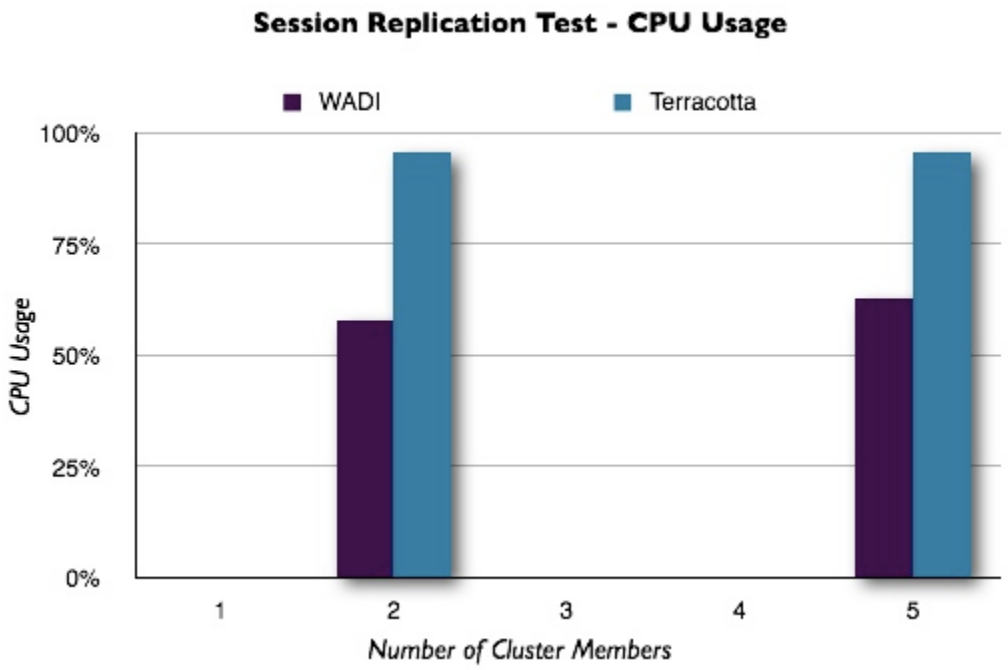
Raw Data

1. of nodes	Average Response Time - WADI	Average Response Time - Terracotta
1	N/A	N/A
2	4.77	6.33
3	Not Measured	Not Measured
4	Not Measured	Not Measured

5	4.87	6.36

CPU Usage

Graph



Raw Data

1. of nodes	**Average CPU Usage - WADI**	**Average CPU Usage - Terracotta**
1	N/A	N/A
2	58	96
3	Not Measured	Not Measured
4	Not Measured	Not Measured
5	63	96

Observations

- Once again, the cost of keeping one session replica remains constant while the number of nodes in the cluster increases for WADI.
- GETs served by Jetty instances clustered by WADI are about 24% faster than Terracotta - for the considered scenario.
- WADI is significantly less CPU intensive than Terracotta - for the considered scenario. As a matter of fact, Terracotta used the full CPU resources.

Conclusion

The effectiveness of the design and implementation of WADI's distributed session lookup engine and replication engine is further comforted by the observed average response times and scalability characteristics.

For the considered scenarios, WADI performs better than Terracotta, which is not really surprising as the size of the state stored in HTTP session is too small. Indeed, Terracotta implements a sophisticated state tracking mechanism, which payoffs more and more as the size of sessions increases. It appears that one of the drawback of this advanced tracking mechanism is that it is more of an overhead than an actual benefit for small size sessions, which should be the most common case for online solutions. In order to efficiently cluster the few online solutions having a rather big session size, WADI offers an alternative replication mechanism, which only replicates field updates or method executions against replicas.

As of this writing, WADI's future directions are:

- offering of an efficient and robust caching solution leveraging the core infrastructure services validated by the tests of this report; and
- more complete integration with Geronimo.