Operations Guide

About Terracotta Documentation

This documentation is about Terracotta DSO, an advanced distributed-computing technology aimed at meeting special clustering requirements.

Terracotta products without the overhead and complexity of DSO meet the needs of almost all use cases and clustering requirements. To learn how to migrate from Terracotta DSO to standard Terracotta products, see Migrating From Terracotta DSO. To find documentation on non-DSO (standard) Terracotta products, see Terracotta Documentation. Terracotta release information, such as release notes and platform compatibility, is found in Product Information.

Release: 3.6
Publish Date: November, 2011 **Documentation Archive »**

Operations Guide

Introduction

Living successfully with a Terracotta deployment, as with any production deployment, requires developing and rigorously adhering to an operating plan. This plan should include a runbook that describes what steps to take for all routine operations on your cluster as well as how to monitor the health of your cluster and take action against unplanned events, such as hardware failures, load spikes, and the like.

If you want support for your Terracotta deployment, contact us about our Enterprise Support Options.

Set Up Monitoring

Terracotta provides a wealth of information about the current state of your cluster. Before deployment, you should have a plan for what to monitor and how you are going to integrate it into your network operations center.

There are a few different ways to retrieve this information for your purposes.

The Terracotta Developer Console

The Terracotta Developer Console provides a graphical view of all of the runtime information available from Terracotta. The Developer Console is especially useful during development and testing.

Enterprise versions of Terracotta also include the Terracotta Operations Center, a GUI operator's console offering features such as backups of shared data, client disconnect, and server shutdown controls. To learn more about the many benefits of an enterprise version of Terracotta, see our [Enterprise Products|enterprise:Products].

The Terracotta JMX Interface

The Terracotta JMX Interface can be used to retrieve all of the data presented by the Terracotta Developer Console and Operations Center. Additionally, because it uses the standard the JMX protocol, it can be integrated into your existing monitoring and management infrastructure. Likewise, it can be used to create alerts when values cross certain tolerance thresholds.

The Terracotta Cluster Statistics Recorder and Snapshot Visualization Tool

In addition to the runtime statistics offered via JMX and the Developer Console, Terracotta offers highly detailed statistics that may be recorded, stored into a database, and viewed using the Terracotta Snapshot Visualization Tool (SVT). Because gathering these more detailed statistics can add significant load to the Terracotta server array, they are not available as always-on runtime statistics. You should, therefore, be judicious in your gathering of these statistics.

For more information on using the Terracotta Cluster Statistics Recorder and the Snapshot Visualization Guide, see the Platform Statistics Recorder Guide and the Developer Console.

Baselines And Tolerances: Understanding How Your Application Behaves

During your performance and destructive testing phase, you should be gathering baselines for the relevant statistics that you are going to monitor. You should also establish tolerance bands for them so that you know when the health of your cluster is degrading. Once in production, you should maintain a regular schedule of baselining your application and re-evaluating your tolerances and adjusting your runbook accordingly. You must also do this during major software releases, as well.

Baselining and tolerance setting is not something you can just do once and forget about. It is an ongoing process of adapting to changing workload characteristics and changing software. The better you understand how your application behaves under a wide range of conditions, the better equipped you will be to handle critical situations as they arise.

Create A Dashboard View Of Your Runtime Statistics

You should create a dashboard view of all of the most important statistics that indicate the health of your cluster. This will let you see at a glance when your application health starts to operate outside tolerances.

Monitoring for Issues

The following section describes various issues that may arise during the ongoing operation of your cluster, how to monitor and alert for them, and what corrective actions to take. You can use this section as the starting point for your cluster operations runbook.

Issue: Disk Full

Description

If a file system that a Terracotta server instance writes to is full, that server instance will throw an exception. If the file system in question is only used for logging, logging will stop, but the cluster will continue to operate. If the file system in question contains the Terracotta server instance's data store (the Sleepycat database), the server instance will fail. In this case, a Terracotta cluster with more than one server instance will continue operation.

Symptoms/Manifestation

The system logs (for example, /var/log/messages or /var/adm/messages) will contain a "disk full" message. If the file system containing a Terracotta server instance's data store is full, the server log will contain an exception similar to the following:

```
TerracottaServerLog: [WorkerThread(commit_changes_stage,2)] ERROR com.tc.server.TCServerMain -
Thread:Thread[WorkerThread(commit_changes_stage,2),5,TC Thread Group] got an uncaught exception.
calling CallbackOnExitHandlers. Environment invalid because of previous exception:
com.sleepycat.je.RunRecoveryException: (JE 3.2.76) . IOE during Write. Caused by:
java.io.IOException: No space left on device
```

Monitoring

Monitor disk usage using your standard system monitoring tools.

Suggested Tolerances

- Green: less than 65% full.
- Yellow: between 65% and 85% full.
- Red: greater than 85% full.

Actions

- Ensure no other processes are filling the disk.
- Ensure log-rotation facilities are working properly.
- Increase disk space available to the file system.
- If the Terracotta data store grows continuously, this may indicate either a memory leak in your application or that the Terracotta distributed garbage collector is not cleaning up garbage faster than it is created. See the Tuning Guide for garbage-creation considerations and information on tuning the distributed garbage collector.

Issue: Disk I/O Throughput Degradation

Symptoms/Manifestation

High disk access latency.

Potential Causes

A Terracotta server instance keeps as much clustered object data as possible in a cache in memory. However, if a client requests a shared object not present on heap, a server instance will read that object's data from disk. A high cache miss rate in a server instance will have a negative impact on performance.

Another potential cause is excessive clustered object garbage.

Monitoring

- Disk I/O
- L2 fault from disk

Actions

Increase the heap and the cache size in Terracotta server instances with disk I/O throughput degradation. If excessive garbage is present, try reducing the amount of short-lived clustered objects being created.

Issue: Terracotta Server Instance Low On Memory

Description

For a number of reasons, a Terracotta server instance may run low on memory. See the Tuning Guide for details.

Symptoms/Manifestation

Low memory in a Terracotta server instance is indicated by:

- Long, frequent garbage collection cycles in the Terracotta server JVM (the JVM garbage collector, not the Terracotta Distributed Garbage Collector).
- Pauses during full JVM garbage collection which cause the server instance to stop responding to client requests.
- OutOfMemoryErrors in the Terracotta server log

Monitoring

There are a number of ways to monitor a Terracotta server instance's heap usage:

- Scrape its log files for OutOfMemoryError.
- Use the Terracotta Developer Console or the Terracotta JMX interface to monitor its heap statistics.
- Use the SVT to sample the following statistics:
    - cache-objects-evict-request
    - cache-objects-evicted
    - vm-garbage-collector
    - memory

Actions

You may need to increase the heap size of Terracotta server JVMs that run low on memory. Additionally, you may need to tune the Terracotta virtual memory manager as described in the Tuning Guide.

Issue: Terracotta Server CPU Utilization Too High

Symptoms/Manifestation

If a Terracotta server instance's CPU utilization is high, you may see a dip in the transaction rate reported by that server instance; you may also see a dip in the transaction rate of the cluster as a whole.

Monitoring

Monitor Terracotta server CPU utilization using your standard system monitoring tools or through the Terracotta JMX interface. Make sure you monitor the utilization of all processors/cores.

Suggested Tolerances

- Green: less than 65% utilization
- Yellow: between 65% and 80% utilization
- Red: greater than 80% utilization

Actions

If not all processors/cores are being utilized equally, you may have a hardware or operating system issue, in which case you should consult your hardware or software manufacturer. If all processors/cores are being utilized equally, you may need to increase the CPU capacity of the Terracotta server machine.

Issue: Terracotta Server CPU Failure

Symptoms/Manifestation

The kernel should panic and the machine will probably go into a reboot sequence. You may also see messages in the operating system logs (for example, /var/log or /var/adm).

Monitoring

Monitor the health of the Terracotta server machine's CPU using your standard system monitoring tools.

Actions

In the event of a kernel panic, a Terracotta server instance will fail over to a standby Terracotta server instance called a "hot standby," or simply have its tasks continued by another active server instance in the Terracotta server array. If using a hot standby, the cluster will automatically reconnect to the new active Terracotta server instance and resume normal operations. You can restore your cluster to a high availability configuration by deploying a new hot standby.

Issue: Terracotta Client CPU Failure

Symptoms/Manifestation

Like a failure of a Terracotta server CPU, the kernel should panic and the machine will probably go into a reboot sequence. When it is disconnected from the Terracotta server array, it will be automatically removed from the cluster and any cluster resources it held will be reclaimed.

Monitoring

Monitor the health of a Terracotta client machine's CPU using standard system monitoring tools. If you are using a load balancer, it should also be configured to detect a cluster node failure.

Actions

If you are using a load balancer, the load balancer should automatically detect the failure and rebalance load across the other cluster nodes. To re regain capacity, you should deploy a replacement machine.

Issue: Terracotta Server Array Distributed Garbage Collection Performance Degradation

Description

Under certain conditions, the Terracotta Distributed Garbage Collector's performance may degrade. For more information on this issue, see the Tuning Guide.

Symptoms/Manifestation

If the Terracotta Distributed Garbage Collector (DGC) falls behind, you will see the following:

- Gradual reduction in cluster throughput
- Increased managed object count
- Increased DGC cycle times
- Increase in disk usage by each Terracotta server instance's data store, which collectively is the Terracotta server array's data store.

Monitoring

Use the Terracotta Developer Console or the Terracotta JMX interface to monitor the following:

- managed object count
- DGC total time
- DGC pause time

Suggested Tolerances

The DGC tolerances will vary depending on your application, but as a rule of thumb, DGC cycle times over 30 seconds and pause times over 10 seconds are a cause for concern.

Actions

See the Tuning Guide for information on tuning the DGC.

Issue: Terracotta Server Instance Object Cache Hit Rate Degradation

Description

A Terracotta server instance keeps object data in a memory cache for fast access. If the active set of objects doesn't fit in the server instance's memory cache, you may experience performance degradation.

Symptoms/Manifestation

Poor cache hit rate in the Terracotta server instance is indicated by:

- reduction in cluster throughput
- poor read/write performance given constant disk lookup

Monitoring

- Use the Terracotta Developer Console or the Terracotta JMX interface to monitor the cache miss rate
- Use the SVT to sample l2-faults-from-disk

Actions

Increase the heap and the cache size in Terracotta server instances with disk I/O throughput degradation. If excessive garbage is present, try reducing the amount of short-lived clustered objects being created.

Issue: Application Deadlock

Description

An application concurrency bug may lead to a cluster-wide deadlock. Sometimes, however, what appears to be a deadlock may actually be a very slow operation.

Monitoring

You should have a health check specific to your application that alerts your operations center when there is a degradation in throughput outside certain tolerances. In addition, you may also use the transaction rate form the Terracotta JMX interface or the Terracotta Developer Console as a proxy for cluster throughput.

Actions

- Using the Terracotta Developer Console or the Terracotta JMX interface, take a series of thread dumps 5 seconds apart for 1-2 minutes. If the relevant threads' stack-trace appear stationary, then there is a high likelihood that you have a deadlock
- TO RECOVER: Kill one client JVM at a time - in the best case, the first client JMV restart relieves the cluster of the deadlock and in the worst case the last client JVM restart relieves it of the deadlock.
- To ANALYZE: Based on the thread-dump and the lock profiler in the Terracotta Developer Console, determine where, if in application code, this might be occurring. Also code-review to ensure that locks are being obtained in the same sequence to eliminate possibility of application deadlock. Also consider tools to help deadlock detection.

Cluster Events

Description

The Terracotta JMX interface includes event notifications for cluster events. A node coming online or going offline will fire an event that you can use to monitor the disposition of the cluster. See the JMX Guide for details.

Unknown macro: {HTMLcomment}

Returning To A High-Availability Configuration After Server Failure

TODO

Upgrading To A New Version Of Terracotta or Java

The safest way to upgrade to a new version of Terracotta or JDK is to take the cluster down and upgrade all Terracotta server instances and clients (application servers) before restarting the cluster.

NOTE: THIS ASSUMES THAT THE TERRACOTTA DATA FORMAT STORED IN THE UNDERLYING STORE IS COMPATIBLE BETWEEN THE TWO RELEASES. In the odd-case that it is not, you must remove the data files before restarting the upgraded Terracotta server instance or point the upgraded server instance to an empty directory location in the Terracotta configuration file (by default, `tc-config.xml`).

ALWAYS MAKE SURE THAT YOU TEST THIS PROCEDURE in a staging environment before performing it in production.

Unknown macro: {enterprise-feature}

See the [Terracotta product matrix] for Terracotta products that support rolling (live) upgrades.

Deploying A New Version of Your Application Software

The safest way to upgrade to a new version of your application is to take the down and upgrade all Terracotta clients (application servers). Restart the clients after they are upgraded.

Before attempting to upgrade your application on a Terracotta cluster, be sure to read the sections below on schema changes and object data translation.

Upgrading your cluster to a new version of your application software can be done in a phased approach. One part of the cluster is taken out of service for upgrading, then returned to service while the next part is taken out of service and upgraded, then returned to service, and so on.

Testing upgrades in a staging environment is strongly recommended. The success of upgrading a live cluster depends on how compatible the newer version of your application is with the older version, since the two versions will briefly run at the same time.

The steps are as follows:

1. Take half of the Terracotta client servers (application servers) out of service.
   Application load carried by those machines should fail over to the remainder of the active cluster according to your third-party load balancer or workload router rules.
2. Upgrade your application on the out-of-service machines.
3. Take the remainder of the active Terracotta clients out of service.
4. Simultaneously, return the upgraded Terracotta clients back to service.
5. Upgrade your application on the out-of-service machines.

6. Return the upgraded Terracotta clients back to service.

Class Schema Change Considerations

Terracotta does not use Java serialization to share object data, so upgrading your application code is not subject to the same class-versioning limitations that serialization imposes. However, there are some considerations that you must make when upgrading your Terracotta-enabled code.

Unknown macro: {table}

Unknown macro: {tr}

Unknown macro: {th}

Change Type

Unknown macro: {th}

Current Support

Unknown macro: {th}

Future Support

Unknown macro: {th}

Notes

Unknown macro: {tr}

Unknown macro: {td}

Add, delete, or modify methods

Unknown macro: {td}

Yes

Unknown macro: {td}

Yes

Unknown macro: {td}

Only the object data is stored and manipulated by Terracotta, so changes to methods will have no effect on existing object data. However, you must make sure that method changes are reflected in the Terracotta configuration as necessary.

Unknown macro: {tr}

Unknown macro: {td}

Add a field

Unknown macro: {td}

Yes

Unknown macro: {td}

Yes

Unknown macro: {td}

If you add a field to a class, its value will be the default value for primitives and null for references. You must take care to initialize new fields using Terracotta's "on-load" feature. See the Configuration Guide and Reference for details.

Unknown macro: {tr}

Unknown macro: {td}

Delete a field

Unknown macro: {td}

Yes

Unknown macro: {td}

Yes

Unknown macro: {td}

The field data is preserved for objects created using the old class schema, but will not be visible to code using the new class schema. Objects created using the new class schema will not create data for that field.

Unknown macro: {tr}

Unknown macro: {td}

Modify a field name

Unknown macro: {td}

No

Unknown macro: {td}

Yes

Unknown macro: {td}

Modifying a field name is currently the same as deleting the old field and creating a new field. There is not currently direct support for migrating the field data from the old field name to the new field name, although that can be done with a custom object data translator (see below).

Unknown macro: {tr}

Unknown macro: {td}

Modify a field type

Unknown macro: {td}

Partial

Unknown macro: {td}

Yes

Unknown macro: {td}

This is supported if the old field type can be coerced into the new field type (e.g., 'int' to 'long').

Unknown macro: {tr}

Unknown macro: {td}

Modify an instance field to a static field

Unknown macro: {td}

No

Unknown macro: {td}

Yes

Unknown macro: {td}

Unknown macro: {tr}

Unknown macro: {td}

Modify a static field to an instance field

Unknown macro: {td}

Yes

Unknown macro: {td}

Yes

Unknown macro: {td}

This is equivalent to adding a new field.

Unknown macro: {tr}

Unknown macro: {td}

Change the name of a class

Unknown macro: {td}

No

Unknown macro: {td}

Yes

Unknown macro: {td}

Unknown macro: {tr}

Unknown macro: {td}

Change the type of a root

Unknown macro: {td}

No

Unknown macro: {td}

Yes

Unknown macro: {td}

Custom Object Data Translation

While there is no direct support for modifying the class schema in the Terracotta object database, it is possible to create a custom object data translator that will convert an object graph from one format to another. One possible approach is to write code that walks an existing object graph, copying values from that existing object graph to a new object graph with the new class format. Once the translation has occured, code using the new class format would then use the new object graph. The old object graph will fall out of scope and become garbage collected.

ALWAYS MAKE SURE THAT YOU TEST ALL UPGRADE PROCEDURES in a staging environment before performing them in production.

Unknown macro: {enterprise-feature}

Patching an Installed Version of Terracotta

Your Terracotta installation may need a fix between point releases. [Terracotta software patches] provide a low-impact software upgrade.

Unknown macro: {HTMLcomment}

Upgrading To A New Version Of Java

TODO

Back Up the Cluster Database

The Terracotta cluster has a BerkeleyDB database containing all of the object data for the Terracotta virtual heap as well as some metadata about the state of the cluster.

Unknown macro: {enterprise-feature}

A one-step database backup button is available. The backup captures a full snapshot of the database. See the Terracotta Operations Center for more information on backing up the database.