Deployment 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 »**

Deployment Guide

Introduction

This deployment guide presents the issues that you should consider when
preparing to deploy your Terracotta application. The network and systems provisioning section will help you determine how to properly size your network and computing hardware resources as well as network architecture considerations.

Designing a system architecture and provisioning your network and computing hardware optimally for a Terracotta deployment requires active consideration during the testing phase of your project.

In general, all of the hardware machines in your cluster — application servers (Terracotta clients), and Terracotta servers (active and standby) — should have similar if not identical capacity in terms of CPU, RAM, heap sizing, network hardware and settings, and disk I/O speed. The slowest machine in the cluster can throttle the rest of the cluster over time, and a slower standby (or backup) Terracotta server instance can throttle the active server instance.

For expert help with your Terracotta deployment, contact us at info@terracotta.org about our enterprise support options.

Sizing The Terracotta Client Hardware

Your existing application server hardware should be adequate to serve as Terracotta clustered JVM clients. Your workload partitioning strategy and the number of Terracotta client nodes you deploy will have an effect on the amount of heap allocated to each Terracotta client JVM.

As a general rule, each client JVM should do work on data that fits in its local heap. This good locality-of-reference characteristic will allow your application to operate at memory speed, faulting in object data from a Terracotta server instance only when first joining the cluster or when workload is rebalanced.

Another benefit of good locality-of-referene is that it minimizes the impact of having varying machine configurations across the application side of the cluster. In addition, load balancers can be used to send more work to machines with more resources.

You should also reduce the amount of clustered object data kept resident in the heap of more than one client JVM to a minimum to avoid redundant client JVM heap allocation for that data and to minimize the broadcasting that the Terracotta server must perform as that object data changes.

In exchange for its availability and scalability characteristics, Terracotta introduces some memory and CPU overhead to the application server process. This can be offset by adding more application servers to handle increased capacity requirements.

Determining The Number Of Terracotta Clients

Given that each Terracotta client JVM should work as much as possible with data that fits in heap, the number of Terracotta clients will be affected by the total size of the active clustered data set. You should have enough Terracotta client JVMs so that the entire data set will fit in the sum of the total heap available for clustered object storage across all Terracotta client JVMs. If you partition your workload across Terracotta client JVMs such that there is a minimum amount of overlap in the set of objects kept in heap across clients, you will likewise minimize the total amount of heap required by the entire cluster.

Another factor that affects the number of Terracotta client nodes is how much work each client JVM can handle relative to the overall throughput requirements of your application cluster. During your capacity planning process, you must determine how much throughput each client node can handle as a fraction of the total required throughput. You must also test your application cluster at appropriate scale to make sure that your cluster behaves such that, as client application servers are added, total cluster throughput is increased.

Load Balancing and Workload Partitioning

In order to achieve the good locality of reference mentioned earlier in the section on sizing the Terracotta client hardware such that each client JVM works on data that fits in its local heap and that data does not overlap with other client nodes, you must have some workload partitioning strategy. As a general purpose clustering solution, Terracotta does not ship with a particular workload partitioning strategy in the box. Workload partitioning can be implemented in a number of different ways.

A simple form of such workload partitioning is a layer seven sticky load balancer that is able to route requests for a particular user session to the same Terracotta client application server and is likewise able to balance user sessions evenly across all available application servers. If you are using Terracotta in a user-session context, this sticky load balancing method of workload partitioning is highly recommended. In this scenario, WE STRONGLY DISCOURAGE THE USE OF ROUND-ROBIN OR OTHERWISE NON-STICKY LOAD BALANCING STRATEGIES.

Other forms of workload partitioning may be required for different use cases. Implementing workload partitioning algorithms in Java that can then be made cluster-aware via Terracotta is often a simple way to meet custom workload partitioning requirements.

Sizing Terracotta Server Hardware

Your application will have certain object data size and access pattern characteristics that you must understand in order to properly provision the hardware for each Terracotta server instance. As such, you should do capacity planning as part of your performance testing process.

During your capacity planning process, there are a number of considerations to make when sizing Terracotta server hardware, chief among them CPU, RAM, and Java heap sizing.

Memory Sizing

When determining how much memory to provision, there are two main considerations to take into account: 1) how much Java heap to allocate for a Terracotta server instance; and 2) how much total RAM to allocate for the machine running that instance.

Java Heap Sizing

To decide how much heap to allocate to a Terracotta server instance, you should know the size of the clustered object graph(s) that you expect to create. In the ideal case, each Terracotta client JVM will work on a set of object data that fits in its local heap. See the sections above on sizing the Terracotta client hardware and determining the number of Terracotta client nodes for a discussion of data partitioning and locality of reference considerations that will affect Terracotta client clustered data access patterns.

However, even in a well-partitioned cluster, there will necessarily be events that will cause a Terracotta client to request large amounts of object data from a Terracotta server instance. This will happen when a new Terracotta client node joins the cluster. As a new cluster member, a Terracotta client JVM will have no clustered object data yet in heap and must request any objects it needs from a Terracotta server instance. Also, when a Terracotta client JVM is taken out of service and its workload is rebalanced across the cluster, the other cluster nodes will necessarily request the relevant object data for their new workload from a Terracotta server instance.

To service those data requests as fast as possible, a Terracotta server instance keeps an in-heap cache of object data. Ideally, all of the clustered object data will fit into its heap. If your application latency SLAs allow for occasional disk access to retrieve object data not found in the in-heap server cache, then you can size the server heap smaller than the clustered data set. However, if your application latency SLAs do NOT allow for the latency of disk access, you MUST provision enough heap to fit all of the clustered object data.

Keep in mind that the size of the clustered object data set is determined by the lifecycle of the objects your application creates. If you create large amounts of short-lived objects, those objects will occupy space in the Terracotta server instance's in-heap cache until the Terracotta Distributed Garbage Collector can reclaim them as garbage. Your heap must have enough headroom to accommodate the garbage your application creates as well as the active clustered object data that your application still cares about. Therefore, creating less garbage objects will increase the efficiency of the Terracotta server instance's in-heap cache, decreasing its heap size requirements.

As a final consideration for Terracotta server-instance heap sizing, be aware that very large heaps can lead to JVM garbage collection pauses that will cause the server instance to halt all work and have a negative impact on cluster throughput. We typically do not recommend a heap larger than 6-8 GB. If your requirements are such that you must have all clustered object data in heap, but it will not fit in the heap of the Terracotta server array you've allocated, and partitioning strategies have been exhausted, additional server instances must be provisioned. For more information, contact info@terracottatech.com.

RAM Sizing

Each Terracotta server machine should have enough RAM to accommodate the heap of the Terracotta server JVM. In addition, there should be enough RAM headroom to keep the blocks of the Sleepycat data store files in the host operating system's filesystem cache for fast access.

CPU Sizing

Like memory sizing, sizing the CPU of each Terracotta server machine depends on the access patterns of your application. Ideally, you will provision a server machine dedicated to the Terracotta server instance so that you can manage its service level without the complications of other processes.

A few things to consider when sizing a Terracotta server CPU are the rate of garbage creation and the amount of I/O the clients will perform against the server instance on that CPU. The more garbage your application creates, the more CPU resources will be consumed by the Terracotta Distributed Garbage Collector. Likewise, the higher the object read or write rate between Terracotta client nodes and a Terracotta server instance, the more CPU resources that server instance consumes.

If you have high I/O throughput or extensive server JVM garbage collection or Distributed Garbage Collection, you should use at least a four-core server machine. Of course, only your capacity planning process will tell you for sure what kind of server hardware you will need.

Network Hardware Provisioning

The following are guidelines for sizing your network both between the Terracotta client JVMs and Terracotta server instances, and between the server instances themselves.

- Use gigabit full duplex between Terracotta clients and Terracotta server instances.
- Also use gigabit full duplex between Terracotta server instances if they share the same switches as the client JVM machines. If there is an isolated network between the server instances, then a 10/100 may suffice.
- Make sure the network between the Terracotta server instances is a very low latency network to ensure rapid failover in case of server instance failure.
- Make sure the network interfaces are configured properly for utilizing full capacity and full duplex. It is not unheard of for gigabit interfaces to be misconfigured at the operating system level such that they are operating as 10/100.
- Use redundant NICs and redundant switches with automatic failover (VRRP, HSRP).
- Configure the NIC failover time at the operating system level appropriately; likewise, configure the VRRP failover time appropriately.
- Make sure the dual NICs are configured in failover mode.
- With dual redundant NICs, make sure that each mirrored pair is set up for failover.

Terracotta Server Cluster And Network Architecture For High Availability

Terracotta server instances can be deployed in a high-availability configuration such that if an active Terracotta server instance fails, a hot standby server will automatically take its place and the Terracotta cluster will resume normal operations with no loss of data.

For detailed information on configuring a Terracotta server cluster for high availability, see *Configuring Terracotta For High Availability* in

Error rendering macro 'html'

Notify your Confluence administrator that "Bob Swift Atlassian Add-ons - HTML" requires a valid license. Reason: EXPIRED

.

Next Steps

Operations Guide »