GERONIMO 2.0.2 PERFORMANCE
Understanding the current performance profile

Matt Hogstrom
matt@hogstrom.org
October 2007
Version 0.1

Geronimo 2.0.2 Performance Update
Legal Stuff

Java 2 Enterprise Edition (J2EE), Java 2 Standard Edition (J2SE) and Enterprise Java Beans are trademarks or registered trademarks of Sun Microsystems, Inc. So is Java Enterprise Edition 5.0.

This report characterizes performance using a benchmark sample. Further evaluation is required for suitability of any Application Server in its unique deployment scenario. Don’t go buying $30 million worth of hardware and blame me because you didn’t do any further testing.

This report is not sponsored by, or endorsed by, the Apache Software Foundation. It is simply the work of a committer on the project.

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 2.5 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Derivative works should refer to this document and author.

Geronimo 2.0.2 Performance Update
Table of Contents

*Acknowledgments* 1  
*Summary* 2  
*Introduction* 3  
  *Disclaimers* 3  
*Testing Environment* 5  
  *Hardware* 5  
  *Software* 6  
*Workload* 8  
*Results* 9  
  *Web Tier* 10  
  *EJB Primitives* 12  
  *Trade Scenario* 14  
*Appendix A - Linux Kernel Tuning Parameters* 15
Acknowledgments

I would like to acknowledge the generous contribution of hardware used for this report. Intel Corporation has been immensely helpful in providing the systems for testing. Three of the systems have been used in performance and regression testing of Geronimo and are the foundation of this report. The other system is used for TCK testing and other miscellaneous functions.

On a side note I’d like to thank Marc The’Berge from Intel who has been my primary contact for coordinating the use of these systems. Marc has been a great partner over the years and a good friend. He has been very supportive of the projects goals and been very attentive to supporting us. Thanks bubba ;-)

Belinda, my wife, get lots of credit for being patient with me and my odd habits. I married so well, not sure why she got stuck with me.

I also want to acknowledge Chris Blythe and his contributions to DayTrader. Chris juggles his day job, other projects and seems to continue to find time to work on DayTrader. I appreciate all his hard work and his infinite patience in putting up with me.
Summary

For those that are executive types that simply want to know the answer here is the summary. Apache Geronimo Version 2.0.2 is a significant improvement over Apache Geronimo 1.1.1 in several ways. First, it is a certified Java Enterprise Edition 5.0 server that has a small footprint and easy to use console. It also has improved performance over its previous version in almost every primitive with the exception of some very small web primitives and JNDI lookups. Testing for EJB based primitives show a significant improvement over 1.1.1 (albeit using EJB 3.0 versus EJB 2.1 containers) and was able to execute all modes of operation of the benchmark.

Live heap was very low (65MB) with the application deployed which allows Apache Geronimo to be deployed in applications that require a small memory footprint.

Overall Apache Geronimo 2.0.2 is ready for the most demanding developer and the enterprise.
**Introduction**

The goal of this report is to help users understand the relative performance of the Apache Geronimo Server 2.0.2. Unfortunately, I have not had enough time to generate comparable numbers.

The Performance Target (PT) is a number that represents, in general, the best performance number of those Application Servers tested. There is no hidden AppServer A or AppServer B in the report. Based on my testing I found that in many instances, Open Source Application Servers outperformed Commercial Application Servers while in other areas (specifically the EJB workloads) the Commercial Application Servers outperformed their Open Source counterparts. The Performance Target (PT) is a worst case scenario for the Apache Geronimo Server in that it represents the best number and a bit more in some cases. Bottom line is, these numbers represent a respectable throughput for any application server. That said, performance isn’t everything.

An Application Server could be the best performing one on the market and still not meet many of the Non-Functional Requirements (NFRs) that people look at when making a selection. Price, Performance, Usability, Footprint, among others, are all factors in making that selection. This report comments primarily on the performance component.

**DISCLAIMERS**

In the interest of openness and full disclosure the reader should be aware of the following facts:

First, the author, Matt Hogstrom, works for IBM. I am also a committer on the Apache Geronimo project. I’ve been a performance analyst for several years and was the Performance Architect for the WebSphere Application Server. I participated in ECperf 1.0 (JSR-004) as well as ECperf 1.1 (JSR-131). In addition, I represented IBM to the Standard Performance Evaluation Corporation (SPEC) and participated in the development of SPECjAppServer 2001, 2002 and 2004 in the OSG-Java subcommittee. I also like to SCUBA dive and have three very cool children. I don’t like cats, but we have two now, one more than last time, and I don’t know why.

Second, there are lots of choices out there for Operating System, database, etc. I had to make a decision and I chose SuSE Linux Enterprise and IBM’s DB2. I chose SuSE simply because it is the distribution I’m most familiar with and I like it. I’ve used it on Intel as well as zSeries for several years and its my preferred distribution, and a personal preference. I don’t think the performance of another Linux distribution would be substantially different in terms of a performance result.
As far as databases are concerned I decided to use DB2 for a couple of reasons. First, I use it all the time so I’m familiar with it. I wanted to complete this report quickly and decided to avoid a learning curve of another database. There have been several folks on our user lists that have been using PostgreSQL which would be another interesting result.

As with any testing there are an infinite number of possible combinations of tuning options. I chose to use a set of generally accepted defaults and test with those. One could certainly turn out other results that would be higher or lower when tweaking those options. I’m very interested in options that I may not have tried so please feel free to provide feedback. Please send me e-mail directly at matt@hogstrom.org and I’ll do my best to incorporate your ideas and feedback into future results.

Next, the other significant feature of this report is the choice of a Java Virtual Machine. I decided to use IBM’s 1.5 Version of their JRE for Linux (32-bit SR3). I had originally started testing with the Java 1.5 Virtual Machine from Sun but because it wasn’t clear to me that benchmarking with that VM was clear of legal issues. Also, I used the same VM in the report from last year (at least the same Java Version and it makes comparisons a little less complicated.

Finally, I used a commercial load driver to run the workload. There was no fancy scripting used for the tests and Open Source load drivers were not fast enough for the high volume, low-level, primitives. I’d very much like to use something well known like Grinder or JMeter from the Open Source world but they couldn’t keep up (or maybe I was too boneheaded to figure it out). If someone has some suggestions or donations I’m interested.
Testing Environment

H A R D W A R E

The following block diagram describes the environment used to test the Apache Geronimo Application Server. It is a fairly typical which includes a set of HTP drivers, the System Under Test (SUT) which is where Apache Geronimo ran as well as an external database.

Here is a brief description of the systems and their corresponding hardware / software configurations:

Hardware:

Network:

Switch: DLink DGS-2205 5-port Gigabit switch

Although this is not a commercial switch, during the tests there were no network throughput issues noted.

Driver I

Processor: Dual Quad-core 2.67Ghz w/4MB L2 Cache
Memory: 8GB
Network: Intel PRO/1000 MT Server Adapter

Driver II

Processor: 4 Chip Intel Xeon 7140M: two processing cores, 3.40GHz, 2MB L2 cache (1MB per core), 16MB (unified) L3 cache (Hyper-Threading enabled)
Memory: 8GB
Network: Intel PRO/1000 MT Server Adapter
**System Under Test (SUT)**

**Processor:** Dual Quad-core 2.83Ghz w/6MB L2 Cache  
**Memory:** 8GB  
**Network:** Intel PRO/1000 MT Server Adapter

**Database**

**Processor:** Dual Quad-core 2.67Ghz w/4MB L2 Cache  
**Memory:** 12GB  
**Network:** Intel PRO/1000 MT Server Adapter

The systems above (with the exception of Driver II) were based on the SuperMicro X7DB8+ motherboards. I found that using the on-board Gigabit Ethernet ports I used more CPU than using the Intel PRO Ethernet cards.

**SOFTWARE**

**Operating System:** SuSE Enterprise Linux Enterprise SP1  
2.6.16.46-0.4-smp #1 SMP Mon Apr 2 17:59:08 UTC  
2007 x86_64 x86_64 x86_64 GNU/Linux

I chose SuSE because I’ve used it in the past (cf to Geronimo 2.0.2 Performance Report) and it continues to be my favorite for this kind of work. See Appendix A for Linux kernel tuning parameters.

**JVM:** J2RE 1.5.0 IBM Linux build pxi32dev-20070201 (SR4)

The Virtual machine I got from the IBM DeveloperWorks website. I used the 32-bit VM because the footprint of Apache Geronimo is pretty small (about 65MB with the tested application installed) and I was running with a maximum heap of 1GB. Using a 64-bit VM would have simply decreased performance and decreased available memory due to the longer (64-bit) pointers.

**Database:** Instance “db2inst1” uses “64” bits and DB2 code release ”SQL09013” with level identifier “01040107”.

---

1 Even though this system was capable of running with 8 cores only 4 cores were used. To limit the number of cores the maxcpus=4 parameter was passed to the Linux Kernel at initialization time. This limitation was introduced because the available drivers and database were not capable of handling the load generated by more than 4 cores on the System Under Test.

2 These used the 82546EB chipset which seemed to provide better results using less CPU.
I chose DB2 as the database for two reasons. First, I’ve used it before and I’m familiar with it for tuning and the like. Also, the Geronimo 2.0.2 Performance Report was done with this database and I wanted to maintain consistency between the runs.

**Application Server:** Apache Geronimo 2.0.2 - Tomcat JEE 5 Distribution

- **Version:** 2.0.2
- **Download Location:** [http://geronimo.apache.org/downloads](http://geronimo.apache.org/downloads)
- **Distribution:** JEE 5.0 - Tomcat

- **Download Size:** 56.8MB
- **Unzipped Disk Size:** 70MB
- **Disk space after initial startup:** 111MB

- **Initial Startup time:** 15 seconds
- **After DayTrader 2.0 installed:** 19 seconds

- **Java Command Line:** `./geronimo.sh run`

- **Environment:** `JAVA_OPTS="-Xmx1g -Xms1g"`

Often times people wonder why the disk footprint is larger after the initial startup. During the initial startup the transaction logs are initialized, log files are created and other one-time startup costs are generated.
Workload
The workload used to generate these test results is a Java EE 5.0 application called DayTrader. At the time of the tests DayTrader has not been released. I have provided the EAR used for this test at http://people.apache.org/~hogstrom/performance/2.0.2/ for the curious out there. The plan used for deployment is also available at this location as well as other artifacts used for the test.

DayTrader is an application built around the paradigm of online stock trading. Users log in, look at their portfolio, look up quotes and then buy and sell shares of stock. In addition to the full workload runtime of Stock Trading the application also contains a set of primitives used for performance as well as functional testing. It is important to note that not all elements of DayTrader are for performance testing. Primitives used for messaging are provided for functional testing.

The application is very flexible and can execute in a variety of runtime modes. Each of these runtime modes exercise various elements of the JEE 5.0 architecture. It includes runtime modes for standard Web Tier applications that are based on JSPs and Servlets, JDBC and EJB as well as a WebServices mode for conducting remote operations. It should be noted that the WebServices mode is for functional testing and not performance based comparisons.

For the Apache Geronimo 1.1.1 comparisons DayTrader 1.2 was used. This is because DayTrader 2.0 has been refactored for Java EE 5.0 where DayTrader 1.2 was targeted at J2EE 1.4. The web primitives are the same but the EJB primitives have been re-written for EJB 3.0. Bear this in mind as you review these results.

The application was originally built by IBM as a way to characterize performance for J2EE applications. It was used internally by IBM and donated to Apache Geronimo in 2005. Even though the application is maintained at the Apache Software Foundation it does not depend on Geronimo or in any way favor that Application Server.

More information about the application can be found at http://people.apache.org/~hogstrom/daytrader.
Results

The results of the testing are broken into three categories. The categories are Web Tier, EJB Primitives and Trading Scenario.

The Web Tier contains simple primitives that start with the simplest of primitives, PingServlet, and build to increasingly complex tests up to PingServlet2JNDI. The goal of these tests are to characterize the runtime performance of the various components of the Web Container; these results were based on the Geronimo distribution that includes Tomcat.

The name of the primitives covered in this section all begin with the simple primitive PingServlet. All other primitives are built on this foundation. Since the other primitives are built on PingServlet, in most cases, the PingServlet prefix is not mentioned in the charts. So, for example, where you see 2Include you can assume that the actual primitive name is PingServlet2Include.

The tests were gathered by warming up the workload for a few minutes and then stopping it. The server was allowed to settle down and then the workload was restarted and run for approximately 2 minutes. During this time the server reached a steady state and the throughput was captured and is measured in Requests Per Second. The goal for all tests was to achieve 100% CPU utilization and a steady throughput3.

The results include all (or at least most of the primitives with the exception of functional primitives like PingMDBTopic and such) for Apache Geronimo 2.0.2. Also included are results from Apache Geronimo 1.1.1 running on the same system for a select set of primitives. Despite the fact that both servers are not the same (one being J2EE 1.4 and the other being Java EE 5.0 based) it provides a reference point for performance for most of the primitives for both the Web and EJB tiers. At the time this data was gathered the JBoss has not achieved an operational Java EE 5.0 release that could be compared and there was not enough time to calibrate the GlassFish Server. The subset of primitives was due to lack of time to gather every corresponding datapoint. Other commercial servers generally have some legal restrictions on benchmarking so to abide by their legal requirements they were not tested.

____________

3 This was achievable by all primitives except PingServlet2JDBCWrite. The Application Server was never able to become fully saturated and this appears to be an issue with the high number of database updates. Extensive tuning was not attempted for this primitive.
WEB TIER

The Web Tier was an excellent performer for Apache Geronimo. These results were based on the Tomcat version of the server. For all operations good CPU utilization was obtained (in excess of 98% on the 4-way Application Server).

All of the following results are measured in Requests Per Second which is the number of round trip requests sent from the driver to the tested Application Server. Servlet, HTTPSession1 and PingJSP all show a slight degradation compared to Apache Geronimo 1.1.1. The PingJSP primitive was not run on Geronimo 1.1.1. Despite the fact the numbers are degraded you will see in later primitives that the performance when more meaningful work is done (using Connections, SQL operations, etc.) the overall performance is improved.
Just like the previous primitives we see a slight degradation of the Web-tier primitives. PingServletservlet and PingServlet2JSPEL were not run for Geronimo 1.1.1.

Finally, the JDBC primitives demonstrate that with increasing complexity the slight degradation of the WebTier begins to disappear. PingServlet2JNDI has a few known issues. First, it is a mutable JNDI which is an improvement over 1.1.1. In addition, there is a known performance issue that was not addressed by the time 2.0.2 was released. Expect this performance to increase in a subsequent release.
**EJB Primitives**

The EJB Primitives exercise specific paths through the EJB container. Just like the primitives in the web container, these tests are driven from a servlet which in turn exercises the part of the container the performance engineer wants to focus on. These primitives start with simple Servlet2Session and then work through a variety of increasingly complex entity bean tests that include local and remote interfaces along with Container Managed Persistence including Container Managed Relationships (CMRs).

The EJB story is significantly improved over 1.1.1. Notice that the Servlet2Session is dramatically improved. This is a result of more efficient parameter handling during pass-by-value processing. Also, the Servlet2EntityLocal is also vastly improved. There are two reasons for this improvement. The new EJB 3.0 container as well as the use of OpenJPA for persistence. It might be a bit unfair to compare EJB 2.1 and 3.0 persistence together but this data does clearly demonstrate that in Apache Geronimo 2.0.2 the persistence story is vastly improved for those taking advantage of the new container and persistence provided by Apache OpenJPA.

The next set of primitives show an area that needs improvement which is that of complicated SQL operations using CMRs and Collections.
Overall, the performance of EJBs is significantly improved over the preceding versions and specification level.
**Trade Scenario**

These numbers represent the Stock Trading Simulation running in JDBC and EJB Mode. In JDBC mode Servlets, JSPs and JDBC are used to simulate the trading scenarios. Apache DayTrader provides a built in Servlet to simplify testing in this mode of operation. This Servlet is called TradeScenarioServlet and controls the various interactions of the trades. Logins, Quotes, reviewing Profiles and selling holdings are executed in a random fashion based on the cardinality of the database tables. For these tests I ran with a user population of 5000 and a stock quote population of 10000.

Note that the JDBC mode for DayTrader 1.2 was modified slightly. This modification was made to match a change introduced into DayTrader 2.0 that has to due with the Trade Stock Index Market Summary. This function produced a market summary on every transaction and was considered too heavy of a database operation. A modification was made to DayTrader 2.0 so that a market summary was executed once every 20 seconds and the result was cached and returned by all transactions until the next Market Summary Interval. This code was retrofitted onto DayTrader 1.2 so that measurements in this mode would be comparable.

Note also, that there is no EJB mode for Geronimo 1.1.1. That version lacked required functionality that was needed to run the benchmark which is no available in 2.0.2.
Appendix A - Linux Kernel Tuning Parameters

The following parameters were set on each of the Linux systems used in this test. These changes were made after a series of calibration tests. Perhaps not all of them are necessary but this script was run on each of the systems in the test configuration (drivers, SUT and database) for this set of tests. I am including them here for reference.

```bash
#!/bin/bash
#
# ulimit default was 1024
#
ulimit -n 8000
#
# syn backlog defaults to 1024
#
sysctl -w net.ipv4.tcp_max_syn_backlog=4096
#
# max connections defaults to 128
#
sysctl -w net.core.somaxconn=5000
#
# net.core.netdev_max_backlog default is 2500
#
sysctl -w net.core.netdev_max_backlog=5000
#
# Determines the wait between isAlive interval probes default is 75
#
sysctl -w net.ipv4.tcp_keepalive_intvl=15
#
# Number of probes before timing out - default is 9
#
sysctl -w net.ipv4.tcp_keepalive_probes=5
```
Appendix B - Patches to DayTrader 1.2

Index: modules/ejb/src/main/java/org/apache/geronimo/samples/daytrader/TradeConfig.java

--- modules/ejb/src/main/java/org/apache/geronimo/samples/daytrader/TradeConfig.java (revision 586201)
+++ modules/ejb/src/main/java/org/apache/geronimo/samples/daytrader/TradeConfig.java (working copy)
@@ -110,6 +110,15 @@
    private static boolean longRun = true;
    private static boolean publishQuotePriceChange = false;

+    /**
+     * -1 means every operation
+     * 0 means never perform a market summary
+     * > 0 means number of seconds between summaries. These will be
+     * synchronized so only one transaction in this period will create a summary and
+     * will cache its results.
+     */
+    private static int  marketSummaryInterval = 20;
+
+    /*
+     * Penny stocks is a problem where the random price change factor gets a stock
+     * down to $.01. In this case trade jumpstarts the price back to $6.00 to
+     */
@@ -861,5 +870,12 @@
    return publishQuotePriceChange;
  }
+
+    public static void setMarketSummaryInterval(int seconds) {
+        TradeConfig.marketSummaryInterval = seconds;
+    }
+    public static int getMarketSummaryInterval() {
+        return TradeConfig.marketSummaryInterval;
+    }
+
}

Index: modules/ejb/src/main/java/org/apache/geronimo/samples/daytrader/TradeAction.java

--- modules/ejb/src/main/java/org/apache/geronimo/samples/daytrader/TradeAction.java (revision 586201)
+++ modules/ejb/src/main/java/org/apache/geronimo/samples/daytrader/TradeAction.java (working copy)
@@ -38,6 +38,11 @@
  */
    private static final Integer marketSummaryLock = new Integer(0);
    private static long nextMarketSummary = System.currentTimeMillis();
    private static MarketSummaryDataBean cachedMSDB = MarketSummaryDataBean.getRandomInstance();
    private TradeServices trade = null;
    private static TradeHome tradeHome = null;
    private static TradeJDBCHome tradeJDBCHome = null;
    }
Appendix B - Patches to DayTrader 1.2

```java
/**
 * Compute and return a snapshot of the current market conditions. This
 * includes the TSIA - an index of the price of the top 100 Trade stock
 * quotes. The openTSIA (the index at the open) The volume of shares traded,
 * Top Stocks gain and loss
 */
public MarketSummaryDataBean getMarketSummary() throws Exception {
  /**
   * Market Summary is inherently a heavy database operation. For servers that have a caching
   * story this is a great place to cache data that is good for a period of time. In order to
   * provide a flexible framework for this we allow the market summary operation to be
   * invoked on every transaction, time delayed or never. This is configurable in the
   * configuration panel.
   */
  @return An instance of the market summary
  */
  public MarketSummaryDataBean getMarketSummary() throws Exception {
    if (Log.doActionTrace()) {
      Log.trace("TradeAction:getMarketSummary()");
    }
    if (TradeConfig.getMarketSummaryInterval() == 0) return getMarketSummaryInternal();
    if (TradeConfig.getMarketSummaryInterval() < 0) return cachedMSDB;
    
    /**
     * This is a little funky. If it's time to fetch a new Market summary then we'll
     * synchronize
     * access to make sure only one requester does it. Others will merely return the old
     * copy until
     * the new MarketSummary has been executed.
     */
    long currentTime = System.currentTimeMillis();
    
    if (currentTime > nextMarketSummary) {
      long oldNextMarketSummary = nextMarketSummary;
      boolean fetch = false;
      synchronized (marketSummaryLock) {
        /**
         * Is it still ahead or did we miss lose the race? If we lost then let's get
         * of here as the work has already been done.
         */
        if (oldNextMarketSummary == nextMarketSummary) {
          fetch = true;
          nextMarketSummary += TradeConfig.getMarketSummaryInterval()*1000;
        }
        
        /**
         * If the server has been idle for a while then it's possible that
         * could be way off. Rather than try and play catch up we'll simply get in
         * sync with the
         * current time + the interval.
         */
        if (nextMarketSummary < currentTime) {
          nextMarketSummary = currentTime + TradeConfig.getMarketSummaryInterval()
          ();
        }
    }
    return getMarketSummaryInternal();
  }
}
```
/**
   * If we're the lucky one then let's update the MarketSummary
   */
   if (fetch) {
      cachedMSDB = getMarketSummaryInternal();
   }
   return cachedMSDB;
}

/**
   * Compute and return a snapshot of the current market conditions This
   * includes the TSIA - an index of the price of the top 100 Trade stock
   * quotes The openTSIA (the index at the open) The volume of shares traded,
   * Top Stocks gain and loss
   * @return A snapshot of the current market summary
   */
   public MarketSummaryDataBean getMarketSummaryInternal() throws Exception {
      if (Log.doActionTrace()) {
         Log.trace("TradeAction:getMarketSummary()");
      }