

PERFORMANCE COMPARISON REPORT

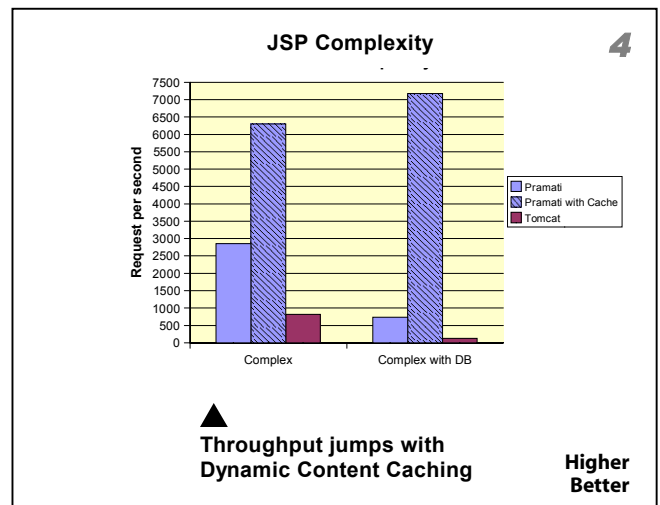
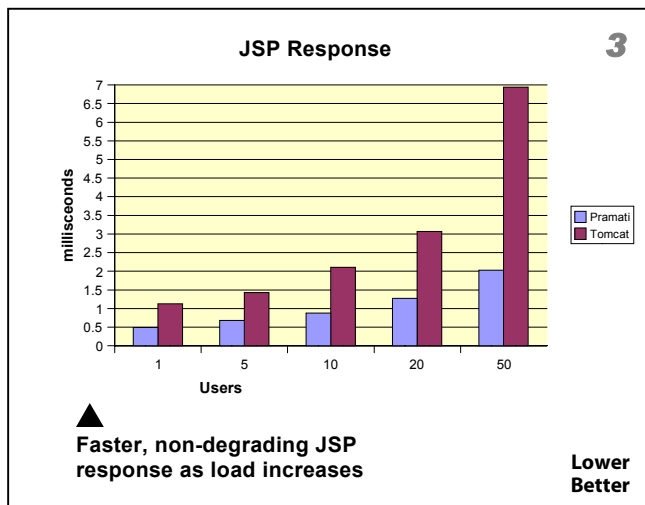
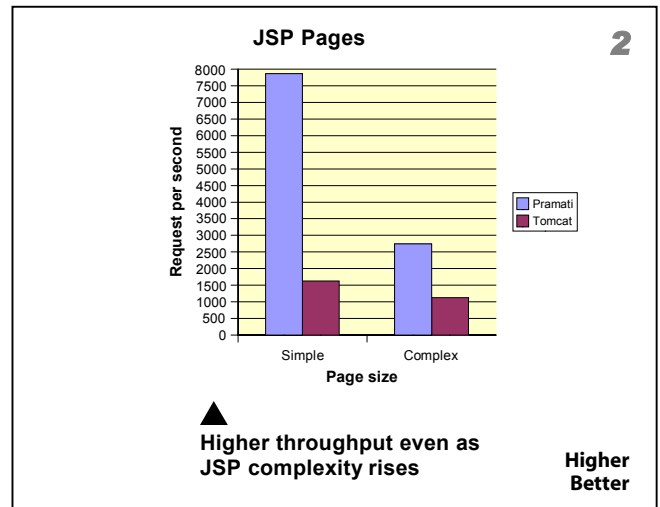
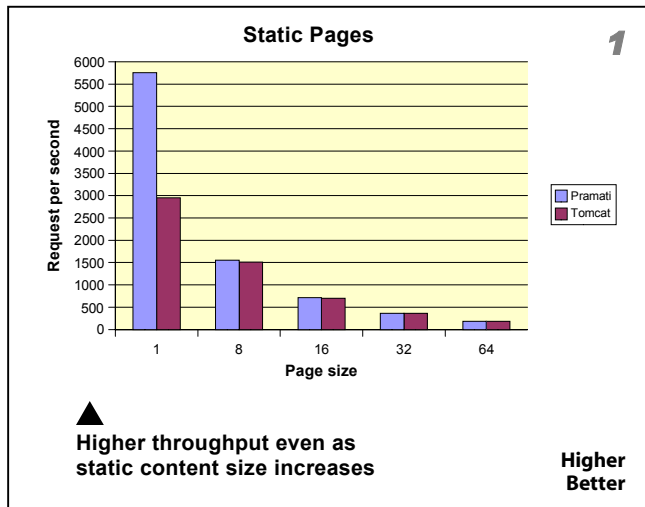
Pramati Server 3.5 and Tomcat 4.0.3 running on Linux 7.3 with MySQL 4.0.1 database

Sridhar Bhamidi, Performance Engineering

Test results show that the Pramati Server 3.5 serves static content twice as fast as Tomcat on Linux. Dynamic content, involving JSP pages with latency, is served up to four times faster than Tomcat. Performance is the single most important consideration for web applications involving dynamic content, JSP pages and databases. Pramati Server 3.5 incorporates several unique performance enhancements and features including the unique Dynamic Content Cache that boosted performance to 16 times Tomcat in some cases.

Operating System	JDK	Database	Processors	RAM	Hardware Vendor
RedHat Linux 7.3	1.4.2	MySQL 4.0.1	2-CPU Xeon	2 GB	Dell PowerEdge 4600

Summary graphs



Introduction

This report is intended to evaluate the Servlet performance of standalone configurations of Pramati Server 3.5 and Tomcat 4.0.3. This is expected to be applicable to small-to-medium projects that are deploying applications in departmental environments.

The approach used to compare performance was to identify the various parts of a typical web application and characterize each type of component separately. Controlled test environment was used to collect the performance results.

Application Server Setup

Default configurations were used for both Pramati 3.5 and Tomcat 4.0.3 with the debug mode, event mode and web logging turned off in Pramati Server and Tomcat.

Load Generator (WAST) Setup

Microsoft Web Application Stress Tool (WAST) was used as a client driver to pump requests. The tool allows configuring many parallel threads. This feature was used to simulate multiple virtual users.

Test Cases

Four scenarios were used to comprehensively gauge the performances of the two Web Containers:

Case 1: Throughput scaling with size of static content (images of 8k, 16k, 32k, 64k)

The WAS tool simulated 50 users accessing 8k, 16k, 32k, and 64k JPEG format images, one at a time for a period of 60 seconds each. The requests per second (throughput) from both servers were measured for each case. See Graph 1.

Case 2: Throughput with complexity of dynamic page (simple and complex JSP)

Two JSP pages were used in this case, one simple and another complex. The simple JSP prints HelloWorld. The complex JSP uses includes, Taglibs, filters and Java processing.

These were accessed one at a time by the WAS tool simulating 10 users for a period of 60 seconds. The throughput was measured in each case. See Graph 2.

Case 3: Response scaling with rising number of users

The WAS tool simulated multiple number of virtual users (1, 5, 10, 20, 50 and 100) and accessed a simple HelloWorld JSP. The response times (in milliseconds) were measured. A delay of 200 milliseconds was configured in WAS tool to prevent system under test from overloading and affecting accuracy of measurements. See Graph 3.

Case 4: Throughput scaling with two complex JSP pages, one using DB (Dynamic Content Cache used)

WAS tool simulates 10 users accessing the complex JSP and complex JSP with DB call. Pramati was tested with and without Dynamic Content Cache feature turned on. Tomcat does not provide Dynamic Content Cache support. The throughput was measured. See Graph 4.

Why Pramati Server performs better

Pramati Server is built on a highly extensible Services Framework with clearly separated services each with a highly optimized implementation.

- Smart request processing with optimized header parsing and stacked output writers.
- Optimized Session invalidation with efficient data structured and Minimal processing overheads
- Single-go write on socket
- Security and Filter processor that defer any processing until (and only as much as) actually needed.
- Content compression for optimal network usage
- Revolutionary 'Extreme threading' Minimizes CPU contentions by throttling various execution paths of the server based on the resource requirements in those paths.
- Object Caches—Task Caching, Request Caching and Response caching
- Metadata caching—information for virtual hosts, WAR applications cached, dynamic pages cached
- TagLib module cache
- Object Pools—worker thread pool, JSP writer, page context and JSP factory pooling
- Byte array and char array pooling
- Reuse of writers—Print, ServletOutputStream, PramatiByteArrayOutputStream
- Generic Recycle Factory Framework to enable object reuse.

SYSTEM CONFIGURATION

J2EE Application Server (1 system)

Hardware	Dell PowerEdge 4600
Processors (2)	Intel Xeon 2200 MHz
Memory (MB)	2048 RAM
OS Name	RedHat Linux 7.3 SMP
Disks	1x18Gb 10K RPM Ultra-160
JDK	Sun J2SDK 1.4.2

Client System (1 system)

Hardware	Dell PowerEdge 1400SC
Processor	Intel Pentium III 1300 MHz
Memory (MB)	1024
OS Name	Microsoft Windows 2000 Server
Disks	3x18Gb 10K RPM U160

Database Server (1 system)

Hardware	IBM NetVista
Processor	Pentium III 800 MHz
Memory (MB)	256
OS Name	Redhat Linux 9.0
Disks	1x10GB
Database Vendor	MySQL AB
Database Version	4.0.1 Standard
Database Driver	MySQL Connector Java 3.0.8 stable