# Chapter 8.9
# Web Application Server Clustering with Distributed Java Virtual Machine[1]

**King Tin Lam**
*The University of Hong Kong, Hong Kong*

**Cho-Li Wang**
*The University of Hong Kong, Hong Kong*

## ABSTRACT

Web application servers, being today's enterprise application backbone, have warranted a wealth of J2EE-based clustering technologies. Most of them however need complex configurations and excessive programming effort to retrofit applications for cluster-aware execution. This chapter proposes a clustering approach based on distributed Java virtual machine (DJVM). A DJVM is a collection of extended JVMs that enables parallel execution of a multithreaded Java application over a cluster. A DJVM achieves transparent clustering and resource virtualization, extolling the virtue of single-system-image (SSI). The authors evaluate this approach through porting Apache Tomcat to our JESSICA2 DJVM and identify scalability issues arising from fine-grain object sharing coupled with intensive synchronizations among distributed threads. By leveraging relaxed cache coherence protocols, we are able to conquer the scalability barriers and harness the power of our DJVM's global object space design to significantly outstrip existing clustering techniques for cache-centric web applications.

DOI: 10.4018/978-1-60566-661-7.ch028

## INTRODUCTION

Scaling applications in web server environment is a fundamental requisite for continued growth of e-business, and is also a pressing challenge to most web architects when designing large-scale enterprise systems. Following the success of the Java 2 Platform, Enterprise Edition (J2EE), the J2EE world has developed an alphabet soup of APIs (JNDI, JMS, EJB, etc) that programmers would need to slurp down if they are to cluster their web applications. However, comprehending the bunch of these APIs and the clustering technologies shipped with J2EE server products is practically daunting for even those experienced programmers. Besides the extra configuration and setup time, intrusive application rework is usually required for the web applications to behave correctly in the cluster environment. Therefore, there is still much room for researchers to contribute improved clustering solutions for web applications.

In this chapter, we introduce a generic and easy-to-use web application server clustering approach coming out from the latest research in distributed Java virtual machines. A *Distributed Java Virtual Machine (DJVM)* fulfills the functions of a standard JVM in a distributed environment, such as clusters. It consists of a set of JVM instances spanning multiple cluster nodes that work cooperatively to support parallel execution of a multithreaded Java application. The Java threads created within one program can be distributed to different nodes and perform concurrently to exploit higher execution parallelism. The DJVM abstracts away the low-level clustering decisions and hides the physical boundaries across the cluster nodes from the application layer. All available resources in the distributed environment, such as memory, I/O and network bandwidth can be shared among distributed threads for solving more challenging problems. The design of DJVM adheres to the standard JVM specification, so ideally all applications that follow the original Java multithreaded programming model on a single machine can now be clustered across multiple servers in a virtually effortless manner.

In the past, various efforts have been conducted in extending JVM to support transparent and parallel execution of multithreaded Java programs on a cluster of computers. Among them, Hyperion (Antoniu et al., 2001) and Jackal (Veldema et al., 2001) compile multithreaded Java programs directly into distributed applications in native code, while Java/DSM (Yu & Cox, 1997), cJVM (Aridor, Factor, & Teperman, 1999), and JESSICA (Ma, Wang, & Lau, 2000) modify the underlying JVM kernel to support cluster-wide thread execution. These DJVM prototypes debut as proven parallel execution engines for high-performance scientific computing over the last few years. Nevertheless, their leverage to clustering real-life applications with commercial server workloads has not been well-studied.

We strive to bridge this gap by presenting our experience in porting the Apache Tomcat web application server on a DJVM called JESSICA2. A wide spectrum of web application benchmarks modeling stock quotes, online bookstore and SOAP-based B2B e-commerce are used to evaluate the clustering approach using DJVMs. We observe that the highly-threaded execution of Tomcat involves enormous fine-grain object accesses to Java collection classes such as hash tables all over the request handling cycles. This presents the key hurdles to scalability when the thread-safe object read/write operations and the associated synchronizations are performed in a cluster environment. To overcome this issue, we employ a home-based hybrid cache coherence protocol to support object sharing among the distributed threads. For cache-centric applications that cache hot and heavyweight web objects at the application-level, we find that by using JESSICA2, addition of nodes can grow application cache hits linearly, significantly outperforming the share-nothing approach using web server load balancing plug-in. This is attributed to our *global object space (GOS)* architecture that virtualizes network-wide memory resources for caching the application data as a unified dataset for global access by all threads. Clustering HTTP sessions over the GOS enables effortless cluster-wide session management and leads to a more balanced load distribution across servers than the traditional sticky-session request scheduling. Our coherence protocol also scales better than the session replication protocols