Load Balancing (LB) in OLE

Table of Contents

Introduction
LB functionality built into Web Server Implementations
    Apache Tomcat Native Connector
        Basic Load Balancer
        Advanced Load Balancer
    Apache HTTP mod_proxy
        Request Counting
        Weighted Traffic Counting
        Pending Request Counting

Introduction

Load Balancing is an important way in which tasks can be distributed across multiple computers. The primary reason to implement load balancing is to reduce workloads. This is achieved by distributing the workload to multiple computers rather than processing them all in a single computer. This helps in easy scaling up of an application to handle much larger datasets.

Another important reason to implement load balancing is to provide redundancy. It helps when there is a hardware failure, a network outage or even a planned scheduled shutdown for upgrades. The load balancer would be able to redirect requests to other active servers bypassing the idle server.

OLE with its Service-oriented architecture (SOA) is well positioned to make use of load balancing. The document would strive to provide pointers on how load balancing can be achieved while deploying OLE.

Load balancing can be achieved by various methods. There are both hardware and software solutions to load balancing. Hardware solutions based on switching technology are provided by various vendors (Cisco, Radware, etc.) and come with customized software which helps in balancing load through configurations done through a management console. Zeus Extensible Traffic Management - Load Balancer (ZXTM-LB, now owned by Riverbed Technology©) is one such hardware which had been tested earlier by Kuali.

Load balancing software packages were shipped as part of operating systems such as Windows NT/2000 and Red Hat Linux. There are also packages by independent vendors such as Resonate's Central Dispatch™ which helps in load balancing.

Many web server implementations have built-in software load balancing functionality. Apache Tomcat, the open-source web server of choice to OLE, recommends two ways in which load balancing can be achieved. One is through a native connector and the other involves using another web server, the Apache Http Server. More details of these would be covered going forward as these are the more commonly used.

It is imperative to externalize persistent dataset so that multiple instances of the server cluster write data to the same instance of database (even RAID configurations virtualize to expose single instance). OLE uses a relational database management system (MySQL, Oracle, etc) to store relational data. It also uses Apache Solr to index, store and retrieve bibliographic data to aid in faster search capabilities typically required by libraries.

It is also a better practice to host any middleware that the web application uses in a separate server so the resources serving the web application are given a respite from running the middleware. OLE uses the Kuali Rice middleware for identity management (KIM), notification

(KEN), workflow (KEW) and service (KSB). While it makes sense to go for a standalone middleware (Rice), in certain cases where a strong coupling is required or when the middleware is not shared by multiple applications, it is not imperative to make the middleware standalone. Different ways in which Kuali Rice can be configured is available here.

LB functionality built into Web Server Implementations

A load balancer is expected to manage multiple instances of a server cluster to handle incoming requests. The management would include the following

- Instantiating the multiple instances in the server cluster.
- Information on the load-balancing factor (strength of an instance) to perform weighted load balancing.
- Session stickiness so that the response to request from a particular session is maintained in a single instance.
- Ability to identify failed instances.
- Reports on statuses and other metrics.

Apache Tomcat recommends two ways to go about load balancing - a native connector in Apache Tomcat Server or using mod_proxy with Apache HTTP Server.

Apache Tomcat Native Connector

A load balancer is considered a worker that is responsible for the management of other 'real' workers. The native connector provides a basic and an advanced load balancer through configurations.

Basic Load Balancer

A basic load balancer helps to balance workers and maintain sticky sessions. The balancing of workers is dependent on the individual strength of the members which is configured as *lbfactor*. Higher *lbfactor* leads to more requests and vice versa.

Advanced Load Balancer

The advanced load balancer supports complex failover configurations. Configurations can be made so that a request would be routed to the member at the least distance. It also provides more detailed configurations to handle sessions.

More information on how to set a load balancer and associate configurations are available here.

Apache HTTP mod_proxy

This would require an Apache HTTP server to be hosted which would handle requests, process them for load balancing and send them to the various instances of Apache Tomcat servers hosting the web application.

There are three load balancer algorithms currently available for use - Request Counting, Weighted Traffic Counting, Pending Request Counting. Session stickiness is provided through cookies and URL encoding.

Request Counting

The request counting method distributes the request among various workers based on the configured *lbfactor* and *lbstatus.* The *lbfactor* gives the worker's quota and the *lbstatus* tells how urgently the worker needs work. The worker with the highest *lbstatus* is first selected and the status reduced by the quota distributed to all workers.

Weighted Traffic Counting

Weighted traffic counting is similar to request counting and differs only in *lbfactor* is the traffic in bytes that a worker needs to handle.

Pending Request Counting

The scheduler keeps account of how many request each worker is assigned at any given time and a new request is automatically assigned to the worker with the least number of active requests.

More details and associated configurations are explained here.