

General Scalability Plan for Enterprise Deployment A White Paper

This document is a property of Analytica India Pvt. Ltd. No part of this document may be copied or reproduced in any form or by any means, electronic, mechanical or otherwise without the prior

Introduction

The previous section ensures the design of a system that can either operate, or can be made to operate in a variable tier environment. However, the actual deployment depends primarily on the customer requirements. In most cases, there are two main constraints that have to be balanced against each other when deciding on the deployment architecture for a system.

1. The expected load (number of simultaneous users) that the customer expects to have. In most cases, to be conservative, a 100% “burst” capability should be built in (i.e., the system should be **able** to handle twice the average load expected by the customer).
2. The customer’s hardware and software constraints. A two-tier system built using an ordinary web server and a database, requires **at least** one less server than a traditional three tier architecture. Although it is theoretically possible to put all the elements of a two or three tier architecture on a single server, it is not recommended at all for reasons of manageability and performance.

NOTE

A good rule to follow before sizing the servers for a customer, is to first examine the bandwidth. Very often, it is the bandwidth that is the biggest bottleneck to system performance – not the servers. However, bandwidth can be upgraded easily – servers (once operational) cannot. Hence, a pre-design sanity check on bandwidth is essential before the sizing of the server is taken up, but the bandwidth (or lack of it) should **not** be taken as the limiting factor for the design of the system.

The following “rule of thumb” may be used to form an initial idea of the architecture to be used for the implementation.

# Concurrent Users	Architecture Type	Bandwidth	Remarks
< 50	Two tier	512 Kbps	Database should be capable of supporting 5-10 concurrent connections.
50 - 100	Two tier, multiple web servers	1024 Kbps	Database should be capable of supporting 25 simultaneous connections. At this point, database activity should be examined. A two tier architecture should be used only if database activity is low. This architecture is to be preferred for systems that will have a lot of web server activity (heavy amount of images, streaming content, and/or ASP/JSP logic). The database server is likely to be the main processing bottleneck for this architecture, and it’s processing load should be closely monitored.
50 – 100	Three tier, single web server, single middleware server	1024 Kbps	Database should be capable of supporting 5 – 10 simultaneous connections. A three tier architecture should be used if there is a fair amount of database logic involved. This architecture is preferred for systems that have a lot of database processing, and rather straightforward web server presentation (minimal ASP/JSP logic, images etc). It is also advisable to monitor this architecture for bottlenecks on the web server.
> 100	Three tier, single middleware server, multiple web servers.	512 Kbps per 501 users.	The database should be capable of supporting 5 simultaneous connections for every 501 users. All segments of the system – the internet connection, the web, middleware, and the database server, should all be monitored at all times for bottlenecks on processing power.

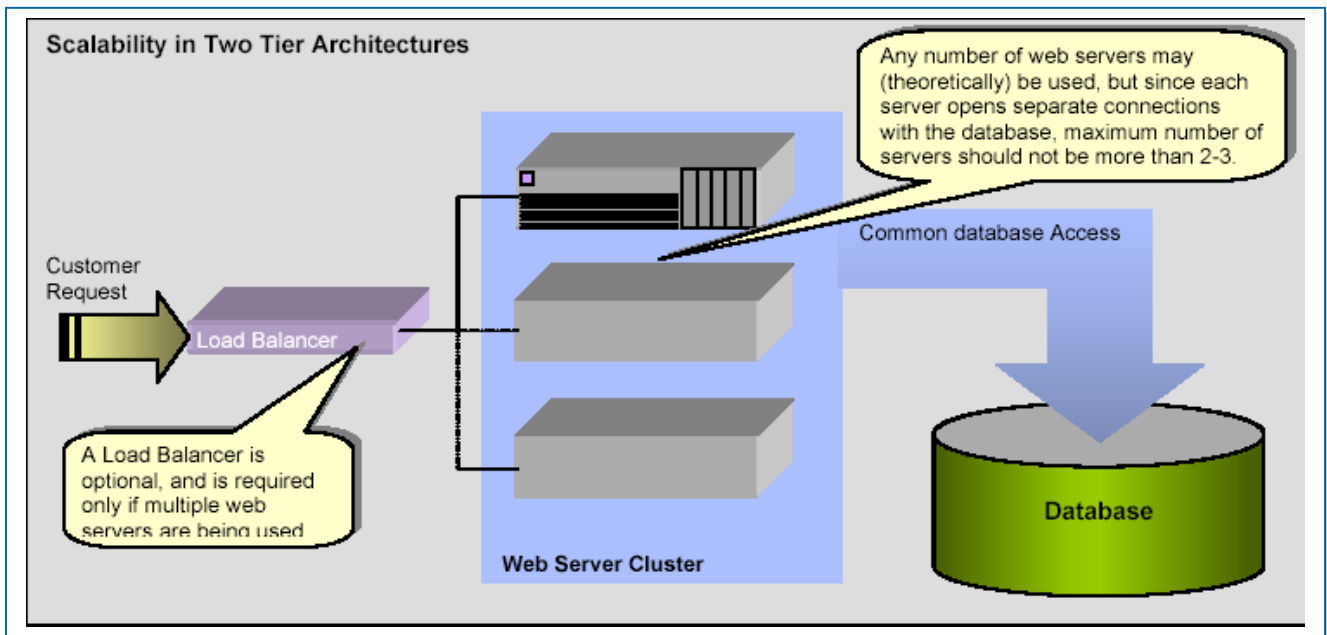
Each of these cases is discussed in more detail in this section.

NOTE

The standard “rule of thumb” for **any** mission critical application is that there should be redundancy. This is especially true of the web server, which handles most of the traffic. So for almost any standard implementation, it is recommended that there be **two** web servers, identically configured and running in parallel, in order to provide a fail-over in the case of a server meltdown. In any enterprise class configuration, almost every element of the system – including even the switches, routers, firewalls, and load balancers - should have a fail-over and/or a backup. This is usually achieved by means of clustering or advanced redundancy techniques. A detailed explanation of these issues can be found in Appendix A of this document.

Two Tier Architecture

This is the simplest type of architecture that is suitable for clients with very simple requirements, low customer volume, and low overall system complexity. This is also the preferred architecture for low volume, high performance systems, where saving an extra layer can add to the system performance. However, this needs to be considered very carefully in the context of future scalability.



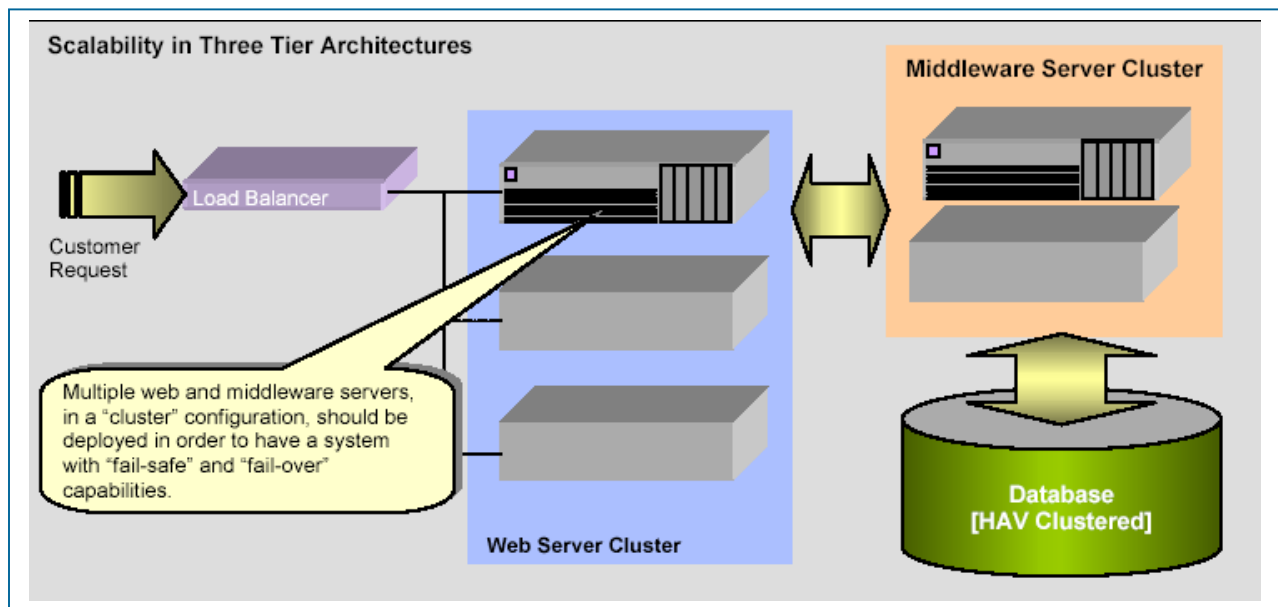
The initial architecture can consist of only one web server, with a direct connection to the database server. There are two ways in which this architecture can be scaled up

1. Adding additional Web Servers as shown in the figure above. Multiple web servers will also require a load balancer to distribute the traffic among all the available web servers. This architecture (multiple web servers) will also add to the overall stability of the website, since other web servers can/will take over when one or more web servers fail. When going with this architecture, it is advisable to configure the system with a load balancer even when one web server is in use, in order to build scalability into the system from the beginning.
2. Moving the entire system to a three-tier architecture. This will require some fundamental re-design of the software. But if the software has been written to conform to the architecture guidelines in section 1 of this document, this exercise should be straightforward. Logic “objects” coded into servlets (or it’s equivalent) will have to be transformed to beans (EJB’s) and called accordingly.

The second alternative is preferred when the system is increasing in complexity as well as volume. If the rate of increase is not very large, then [1] is a good short-term fix to the problem of handling additional volume. Yet another advantage of this “quick-fix” solution is that the additional hardware required will be used even after the architecture transitions to a three-tier configuration, thereby protecting the customer’s investment in hardware and infrastructure.

Three Tier Architecture

This architecture is preferred for systems that are more complex, flexible, and also have more variability in their load handling requirements. Any system for which the front end user interface and/or the business logic is mutable should be designed as a three-tier architecture.



A “Clustering” approach is usually preferred because of its much greater fault tolerance. The aim should be to scale up by clustering servers at each level, as shown in the diagram above. The clustering concept also allows the sizing (both number of servers, as well as the processing power of each server) to be done separately for each level, depending upon the processing power required. This allows, overall, for a much more efficient and cost-effective system.

The diagram above does not show any of the networking equipment required for this system. It must be kept in mind that networking and security equipment, like routers, switches, and firewalls are as prone to failure as servers. Hence for an enterprise class architecture, each of these should be designed with a fail-over in mind. Often, this requires two of everything, connected in parallel so that the second can take over without any manual intervention in the event of a failure.

The key to scaling a three-tier architecture effectively is *monitoring*. The load at each layer or tier of the system should be closely monitored so that bottlenecks can be quickly identified and dealt with. The standard “rules of thumb” for scaling up an three tier architecture are

1. The web server layer should be the first layer to be scaled up. As a design decision, it is better to have a larger number of smaller servers at the web server layer, rather than a fewer number of higher end servers. The addition of many servers is not actually a “clustering” technology, since they should all be configured to act independently when traffic is directed at them.
2. The database server is typically the last layer to be “clustered”, but is also the most critical from the point of view of the system. The clustering technology used here is true clustering, like the IBM “HACMP”, and the Sun “Full Moon” technologies.
3. The middleware servers – usually running Microsoft MTS, BEA Weblogic, IBM WebSphere, or a similar middleware platform, can be ‘pseudo clustered’ using the middleware engine. Most of them have inbuilt technology for operating in a mode with multiple servers in a fail-over configuration.