Performance Driven J2EE Application Design

It’s essential to address the potential performance bottlenecks right at the start of the application design. These bottlenecks are often because the way architecture of the J2EE application is designed. The solution to these bottlenecks lies in a way the code is written, use of better design patterns, memory management, data caching & persistence strategies and the optimal utilization of network resources. The paper discusses the best practices for designing and developing high performance J2EE Web Applications using these solutions.
About the Author

Ajay Ghode
Ajay has been working in the IT arena for more than 7 years and he joined TCS in 1998. He has a Masters in Metallurgy from IISC Bangalore.

Ajay has been associated with development of several J2EE technology projects as Technical Architect and Project Manager. He is working as a consultant in Java Technologies Practice.
Table of Contents

1. Introduction 3
2. The Performance “Lingo” 3
3. Performance Requirements of Common Websites 4
4. J2EE Design Challenges 5
5. The Approach to J2EE Application Design 5
6. Presentation Layer Design Considerations 6
7. Business Component Layer Design Considerations 10
8. Data and Service Access Layer Design Considerations 12
9. Recycle Data and Resources 14
10. Summary 15
Introduction

This is the era of cutting-edge communication technologies where end users are speeding on the broadband internet highways. The competition is already set, where, in a particular business domain several web sites offer the same kind of features, with speed as the only variance. It is just a matter of a few seconds before users abandon your feature-rich yet slow-to-respond web site, and go for the faster one. Speed and Scalability of the web application are the two keys that give e-commerce a boost in the ever-growing and demanding global market place. So Performance is an essential aspect of user acceptance for any software product. This paper is an attempt to identify the potential performance bottlenecks and suggest proven solutions to address it.

The section II explains few major performance terminologies. Section III addresses the performance behavior demanded by some of the common web sites, while section IV compiles the problems faced by end users accessing those web sites. Section V outlines the high level approach to J2EE application design using multi-layer architecture, best practices and proven design patterns. The sections VI to VIII, address the detailed design considerations for each layer in the multi-layered architecture by identifying the performance bottlenecks at that layer and applying solutions by using appropriate design patterns and best practices. Section IX describes the resource recycling methods such as object pooling and data caching and section X, summarizes the analysis.

The Performance “Lingo”

There are numerous ways to define the performance of a web site. But still at large there are certain terms that are inherent means of measuring performance and design considerations. This section throws light on some of these terms:

Peak Load
Peak load refers to the maximum concurrent users sending requests to the web site within some time period. Peak load should always be focus of the design and not the average load \(^1\).

Response Time
Web site response time refers to the time from when the customer initiates a request from his browser until the resulting HTML page returns to the browser. Technically, response time refers to the time between the request and the display of the entire page’s content. However the users perceived response time spans the request to the first appearance of returning page data in the browser \(^1\).

Throughput
Throughput is defined as users served by the application relative to some unit of time. It is a unique metric since it has an upper bound. No matter how many users logged on, the maximum number of requests handled during a specific time interval remains unchanged \(^1\).

Scalability
Both response time and throughput define the performance of a system under a particular load. However, J2EE systems operate under rapidly changing loads, at times coping with an influx of client requests an order of magnitude greater than normal. The readiness of the application to handle such increasing load with addition of resources but without requiring a change in the design is defined as scalability.

\(^1\) Stacy Joins, Ruth Willenborg, Ken Hygh - Performance Analysis for Java Web Sites, Addison Wesley - 2003
Bottleneck
Bottleneck refers the resource contention inside the web site. As the name implies, a bottleneck chokes the flow of traffic through the site as requests contend for one or more limited resources. Technically we find a bottleneck wherever the code processing a request must wait for a resource.

Performance Requirements of Common Websites
The performance requirements of the web sites vary widely depending on the category it falls into, and hence the application design considerations. Therefore the single most important question to ask before beginning a web site design is: what does the web site need to do well? This section explores the performance characteristics of some of the websites in popular business domains.

Financial Sites
The traffic pattern for a financial web site tracks to major market operating hours. Market upheavals result in abnormally large, sustained traffic during the market day. These users repeatedly check the value of their portfolios and review news and research analysis material on companies.

The trading application uses a variety of external resources to prepare the trade, including the customer account database and quote servers. However, the actual trade usually occurs outside the web site in a trading engine. The web site must maintain fast access to this central trading engine and the engine must keep pace with the trading demands. Financial web sites use SSL on a higher percentage of page requests and maintain detailed audit information about the transactions causing extra performance overhead.

Business-to-Business Sites
B2B sites exist to provide services from one company to other companies. The B2B sites do not experience huge swings in traffic and user volumes, as many financial or e-Commerce sites do. A typical day for visitors on some B2B site might include checking the inventory status of a purchase under consideration, placing an order and checking the shipping status of an outstanding order.

A B2B transaction might include processing a large order or receiving a large XML document for processing. Besides, the web site interacts with a variety of backend systems such as order processing, workflow management, inventory and third party systems (such as the shipping company's tracking database) to satisfy user request.

Many B2B sites allow their users to log on to the web site in the morning and retain access to their accounts all day. In such a scenario, if each user’s HTTP session caches a large amount of data over the course of a visit, the memory available to run applications quickly disappears.

B2B systems often interact asynchronously with remote back-end resources using some messaging service. Waiting indefinitely for such asynchronous calls to return often opens the door for “freezing” the web site, if the messaging service begins responding slowly.

E-Commerce Sites
E-Commerce web sites exist to sell products. Users visit these sites to view items, place orders, and use customer service features. Most users browse the online catalog of items or perform searches to view item selections. Only a small percentage of visitors actually make purchases. These web sites compete with other e-Commerce sites as-well-as brick and mortar stores for customers. Rapid page responses give users a better shopping experience and allow them see more pages in less time. Users frustrated by slow response times find another place to shop.

1 Stacy Joins, Ruth Willenborg, Ken Hygh - Performance Analysis for Java Web Sites, Addison Wesley - 2003
Large commercial e-Commerce sites often receive heavy traffic volumes. These sites require good throughput to handle this traffic while maintaining good response times. Web sites creating an HTTP session for each visiting user must manage these sessions carefully to avoid excessive memory consumption.

Any financial transaction on these sites needs to be protected using SSL and this requires more overhead.

**Information Sites**  
Information web sites receive anonymous user traffic. Examples of information sites include news, weather, entertainment, and major sports event web sites. These web sites compete for advertising revenue; the more traffic they handle everyday, the happier the advertisers. Advertising content sometimes comes from third-party sources, complicating the sites performance characteristics. Traffic patterns of the information web sites vary widely depending on the site. However if the site’s information suddenly becomes more interesting, traffic volumes grow dramatically.

**Portal Sites**  
Portal sites offer a personalized “home page” to users. These sites often display a customized view of news, entertainment features, a few selected stocks or other investment data, weather and so on. Such sites may forward requests for page segments to other systems and web sites.

Although portals support many users, the traffic pattern often resemble to those for B2B web site. User log-on is the most expensive operation for many portals. During this process, the web site interacts with security servers (like LDAP) to retrieve information needed to confirm the log-on. The web site creates the users session information during this time and also loads any personalization data for the user. Portals frequently use JavaScript to refresh some of the key pages, generating their own load. Besides some components include applets, which generate load against the web site.

**J2EE Design Challenges**  
Before addressing the J2EE design considerations it is necessary to identify the most common problems faced by the application users. The end-user problems are the results of the various bottlenecks due to design failures and improper tuning of the application. These are the challenges the architect of the J2EE application must take into consideration, which may otherwise lead to the failure of the web site:

1. Pages take too long to load.
2. Search screens take minutes to show results or never return at all.
3. The user interface does not respond, and users click furiously but to no discernible effect.
4. At peak times, the server fails unexpectedly due to “out of memory” errors, locked database tables, and connection timeouts.
5. Previously reliable nightly batch processes become strangely fragile and take longer to run.

**Approach to J2EE Application Design**  
The J2EE application architecture is a combination of user interaction, business logic and the data and resource access components. The design of the application should take into consideration the flexibility, reusability, performance and the scalability of the application. To achieve these objectives a systematic design approach needs to adapted, which may include the logical separation of the components in layers, use of proven design patterns to design these layers and best practices while designing and coding these components. The table-1 summarizes the design recommendations most applicable to common types of websites.

---

Multi Layered Approach
A common framework for J2EE applications is software layering, in which each layer provides functionality for one section of the system. The layers are organized to provide support and base functionality for other layers. Logical layers can ease maintenance, increase flexibility and address deployment concerns. Typical multi layer J2EE architecture is explained in figure-1. These layers and the design considerations for the same are discussed in the subsequent sections.

Fig 1 Multi Layer J2EE Architecture

Use of Proven Design Patterns
As software developers design and build different applications, we come across the same or similar problem domains. This leads us to find a solution for the same/similar problem every time, and we end up “reinventing the wheel”. In the simplest terms, a common solution to such a problem is a design pattern. A design pattern prescribes a proven solution from experienced hands for a recurring design problem. These solutions are generic in nature. Use of such patterns makes the design of an application transparent. All design patterns are reusable and can be adapted to a particular context; this gives developers flexibility. The subsequent sections detail some of the design patterns best fitted for the specific layer.

Use of Best Practices
Developers who may know the Java syntax do not necessarily know how to be effective. Programmers using J2EE technology must learn to go beyond the syntax and understand the best ways to design, architect and implement new J2EE solutions. Without extensive real world J2EE experience. The best course is to learn from the experiences of others. Increasingly, the J2EE platform is proving its strengths and weaknesses in a production setting. You can effectively leverage this experience by understanding the most successful published practices.

3 Derek C. Ashmore - The J2EE Architects Handbook, DVT Press, 2004
4 Vijay S. Ramachandran - Design Patterns for Optimizing the Performance of J2EE Applications, Java Developers Journal, 2001
<table>
<thead>
<tr>
<th>Design Pattern</th>
<th>Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front Controller</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Session Facade</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Data Access</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Service Activator</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Servlet Exit Strategy</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Acquire Static Resource at Initialization</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Small HTTP Session</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Direct JDBC Calls</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>DB Connection Management</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Data Cache</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Value Object</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Object Pools</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Avoid Fine Grained EJB Data Models</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Use of Message Oriented Middleware</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Use Local Interfaces</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Use of Message Oriented Middleware</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Batch Requests</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>

Table 1 Design Recommendations Suitability Matrix

Presentation Layer Design Considerations
The presentation layer is the section of the application responsible for everything end users physically see in the user interface. Various deployment wrappers provide functionality to the presentation layer. J2EE applications produce HTML interfaces by using a combination of static HTML pages and the dynamically generated content via servlets and JSPs.

Performance Bottlenecks
Here we have identified some of the common performance bottlenecks observed at the presentation layer:
1. Exhausted servlet thread pool leading to frozen web site.
2. Complex decision trees in servlets.
3. JSPs containing extensive Java coding.
4. Automatic creation of HTTP session objects in JSPs.
5. Heavily loaded HTTP session objects.
6. Poor concurrent handling of client requests.
7. Synchronization in servlets that avoids multiple execution threads, becoming effectively single-threaded.

Recommended Design Patterns
The request handling mechanism must control and coordinate processing of each user across multiple requests. Use a “Front Controller” pattern as the initial point of contact for handling a request. The controller manages the handling of the request, including invoking security services such as authentication and authorization, delegating business processing, managing the choice of an appropriate view, handling errors, and managing the selection of content creation strategies.

5 Design Patterns section of the J2EE Blueprints Program at http://java.sun.com/blueprints/patterns
Figure-2 illustrates the object interaction diagram for the front controller pattern.

**Best Practices**

**Servlet (Controller)**

*Keep Code Paths Short*
Design the control logic to quickly recognize each request and begin meaningful processing to build a response. Use of decision trees is inefficient, especially if the site handles a lot of different URL and request parameter combinations.

*Provide Exit Strategy For Non-responsive Servlets*
If the servlet calls certain remote system and if that system becomes non-responsive, then the servlet stalls. This makes the thread running that servlet inactive and it never returns to the application servers thread pool. Subsequently, if the thread pool gets depleted, the web site gets frozen and even a single request cannot be serviced. To avoid such freezing of web sites, it is necessary to set a time limit on any request to a remote resource. If within this time limit the resource fails to respond, then the servlet should exit that call.

*Use Servlets To Control The Application Flow*
Do not use the servlets to generate HTML or other output formats, use JSP’s instead. Using servlet strictly as the controller often helps the web application avoid garbage collection issues. The outbound presentation layer usually consists of lots of strings. The JSP's generated by the application server build these strings efficiently and this reduces garbage collection cycles.

*Acquire Static Resources At Initialization*
Considering the performance overheads for JNDI lookups perform any common lookups inside the servlets init () method and then store the results in class variables.
Java Server Pages (JSP)

**Use Tag Library**
Do not place any code inside the JSP’s. Instead use tag libraries and java beans.

**HTTP Session Created By Default**
Some application servers automatically create an HTTP session object for each JSP page if it does not already exist. A session object remains in memory until it times out and is removed by the container. The default time out is often as high as 30 minutes. These session objects take up a lot of heap space in the JVM. If the application being designed does not need HTTP session object created in a JSP page, use a following tag to prevent the web container from creating one automatically.

```
<%@page session="false" %>
```

**JavaBeans and Custom Tags**

**Avoid Bean.instantiate() Method Call**
If the application code creates new java bean using Bean.instantiate() method, the servlet or JSP code checks the hard disk for any persistent bean contents to load, which impacts the performance. To avoid this performance overhead use

```
MyBean bean = new MyBean();
```

**Use A Database If Bean Data Needs To Be Persistent**
This will avoid interaction with hard disk, otherwise thousands of servlet threads will try to dump the bean data on the disk causing a serious bottleneck.

**HTTP Session Sharing And Memory Usage**
HTTP sessions share memory with the running web site applications inside the JVM heap. The J2EE specifications provide no mechanism for controlling the size of an individual HTTP session. The memory utilized by the HTTP session depends on the objects stored in the session including the object trees.

**Preserve Data That Must Persist**
If the application is using HTTP session to store the data that will later go into the database, consider putting them into the database at that instant and keep only the keys in the HTTP session.

**Support A Logout Function**
This lets the users indicate when they have completed their visit and allows the web application to immediately invalidate the users corresponding HTTP session.

**Keep HTTP Sessions Small**
Small HTTP session data reduces the database burden for persistent session storage and is efficient while moving over the network. Besides, small sessions keep the cluster-wide memory requirements manageable.

**Serialize Http Session Data**
The application server serializes the session data while transferring over the network. If the session data does not implement serializable interface, the application server throws an exception. Hence, all the objects and their tress stored in the HTTP session should be serializable.
Avoid Non-Transferable Data
Mark the non-transferable variables in the HTTP session data objects with the transient keyword, so that they will not be serialized. For example, thread handles do not provide any value to another application server instance on failover.

Business Component Layer Design Considerations
Objects in the business logic layer combine data with business rules, constraints and activities. The objects on this layer implement and publish business functionality to the Java objects that could even be on a separate machine. Enterprise Java Beans, RMI services and Web Services are generally used to implement the business functionality. Business objects often use and co-ordinate the activities of the multiple data access objects.

Performance Bottlenecks
Here we have identified some of the common performance bottlenecks observed at the business component layer:
1. Frequent JNDI lookups to obtain EJB Home handles.
2. Number of remote calls to fetch data to server a single HTTP request.
3. Fine grained EJB Data Models.
4. Frequent disk read/write by the container to serialize/de-serialize the object data.

Recommended Design Patterns
Session Façade Pattern
Often there is a lack of uniform business object access strategy that leads to the tight coupling between the clients and the business objects and exposes the business objects to misuse. Performance takes hit, when there are multiple method calls to remote system to satisfy a single request.

Fig 3 Session Façade Pattern
Use a session bean as a façade to encapsulate the complexity of interactions between the business objects participating in a workflow. The Session Facade manages the business objects, and provides a uniform coarse-grained service access layer to clients. This makes the programming interface much simpler as the client calls only one method on a single EJB to perform complex interactions. This reduces the remote method calls, thus improving the performance, if the session bean is in the same container as other EJBs. Figure-3 illustrates the object interaction diagram for the Session Facade pattern.

Best Practices

Enterprise Java Beans

Use Direct JDBC Calls
Consider using direct JDBC calls from within a stateless session bean to perform a multi-row reads of the data, and use an entity bean for updates. This will reduce the expensive finder overheads and provide performance benefits for applications that require a lot of read activities but fewer updates on a complex database table structure.

Avoid Fine Grained EJB Data Models
Even a simple request could end up loading hundreds of EJB’s. Simplify the model and consolidate the data into larger beans. Besides this, use more direct JDBC calls.

Avoid EJB Passivation
If the EJB container runs out of memory while creating new beans, it serializes the state of the stateful bean or passivates the bean, thus creating a performance overhead if the bean needs to be accessed later. To avoid this problem, remove the unused stateful session beans by using Remove() method, and also shorten the time out period of the stateful session beans.

Cache Home’s For EJB’s
Use caching techniques to reduce or eliminate repeated JNDI lookups whenever possible.

Use Of Local Interfaces
It is recommended that you use the local interfaces when the calling clients share the same JVM as the called EJB, since it will eliminate the remote calls. But relying on the local interfaces for performance limits the web site scalability.

Create Value Objects
Create a single representation of all the data that an EJB call needs. By aggregating all of the needed data for a series of remote calls, a single remote call will be executed instead of many remote ones, increasing overall application performance.

Design Patterns section of the J2EE Blueprints Program at http://java.sun.com/blueprints/patterns
Data and Service Access Layer Design Considerations

This layer provides logic in the form of Data Access Objects (DAO) and Service Access Objects to access relational databases and complex legacy system or other middleware services. Besides, DAO’s can manage files, XML documents, and other types of persistent storage as well. The primary reason to separate the data and the service access from the rest of the application is that it’s easier to switch data sources and service resources, and share the DAO’s and service access objects between multiple applications.

Performance Bottlenecks

Here we have identified some of the common performance bottlenecks observed at the data and service access layer.

1. Non-responsive synchronous calls to the asynchronous resources.
2. Depletion of database connection pools because of lost connections.
3. Un-closed database connection pool objects such as statements, result sets etc.
4. Poor cache management.
5. Non-optimized database queries.
6. Frequent writes to directory server.

Recommended Design Patterns

Data Access Objects

Persistent storage APIs vary depending on the product vendor. Other data sources may have APIs that are nonstandard and/or proprietary. Portability of the components is directly affected when specific access mechanisms and APIs are included in the components.

![Data Access Object Pattern](image)

**Fig 4**  Data Access Object Pattern
Use a Data Access Object (DAO) to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and store data. The data source could be a persistent store like an RDBMS, an external service like a B2B exchange, a repository like an LDAP database. The business component that relies on the DAO uses the simpler interface exposed by the DAO for its clients. The DAO completely hides the data source implementation details from its clients. Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes, this pattern allows the DAO to adapt to different storage schemes without affecting its clients or business components. Essentially, the DAO acts as an adapter between the component and the data source. Figure-4 illustrates the object interaction diagram for the Data Access Object pattern.

Service Activator

Sometimes clients need asynchronous processing capabilities from the enterprise beans and other business components that can only provide synchronous access, so that the client can send a request for processing without waiting for the results. Besides clients need to use the message-oriented middleware (MOM) interfaces offered by various vendors. These interfaces are not integrated into EJB server products that are based on the pre-EJB 2.0 specifications.

Use a Service Activator to receive asynchronous client requests and messages. On receiving a message, the Service Activator locates and invokes the necessary business methods on the business service components to fulfill the request asynchronously. Figure-5 illustrates the object interaction diagram for the Service Activator pattern:

![Service Activator Pattern](Fig 5)

---

Design Patterns section of the J2EE Blueprints Program at [http://java.sun.com/blueprints/patterns](http://java.sun.com/blueprints/patterns)
Best Practices

Database Connection Pool Management

Avoid Passing ResultSet To Other Layers
Do not pass result set around to various layers in order to manipulate the data from the database. Create collection objects from the resultset and use them. Then close the resultset, statement and the connection object. Failing to close the statement object before returning the connection to the pool sometimes results in an error for the next application using that connection.

Return Connection Quickly
As soon as the database access operations are over return the connection to the pool, instead of waiting till all the processing is complete. Make sure to close all the database objects and connections in the final clause of the code. Otherwise, if an exception occurs, there will be a database connection leak or a statement may be left open. Too many lost connections in the connection pool may result in a frozen web site.

Use of Vendor Supplied Connection Pools
Connection pools contain complex logic for things such as recognizing and reaping dead connections after a database outage. Often homegrown data sources require considerable tuning and they may not scale as the site grows. Hence, take advantage of tuning expertise built into the vendor supplied implementation.

Batch Requests
Send the number of similar query requests in a single call to the database. This approach allows it to process database requests in a serial fashion at a given time, or as they are received. By doing multiple requests together, only a single connection is required to the database resource and the executing SQL needs to be compiled only once, saving significant cycles. JDBC easily allows for batched requests.

Use of Message-Oriented Middleware’s
Message-oriented middleware’s operate asynchronously, while web applications operate synchronously. Do not try to mask the asynchronous nature of MOMs within the synchronous operation of the web application by placing the web application in a wait state. The indefinite wait time within the web application may lead to disaster, since MOMs do not guarantee delivery within a particular timeframe.

Recycle Data and Resources
Applications waste too much time fetching, creating, or destroying some complex objects or resources. Object pooling and data caching are the two techniques that help in efficiently utilizing the fetched or created resources.

Object Pools
Object pools are used to manage the sharing of objects between multiple client requests. By accessing an existing resource through a pool, the application avoids the creation, destruction and initialization costs. When the request is done with the object, it returns the object to the pool for other client requests to use.

Some of the common characteristics of pools are:

1. Object pools supply a resource on demand.
2. Pools deny the resource if none is available.
3. Pools ensure the validity of the resource.

There are certain considerations that need to be taken into account while designing an object pool:

1. Never let the client request wait indefinitely to obtain the resource handle from the pool. If the resource is not available in a pool, throw a time out exception to the request invoker object.
2. Put the object pool configuration data in an XML file, which will help in scaling up the application.
3. If the object pools are created to serve thread instances, then create a maximum limit for the number of threads. This will prevent stalling of number of threads in case the back-end system becomes unresponsive. In such cases it’s advisable to set the maximum wait time limit for the threads.

**Cache**

A cache is simply another type of pool. Instead of pooling a connection or object, you’re pooling remote data, and placing it closer to the client. A data cache is very similar to a Hashtable. It supports the adding and retrieval of information. A good cache must provide much more, like validation of data, expiration of stale data, and identification and management of infrequently accessed data.

**Summary**

The common performance problems faced by the end users while using most web sites are the results of the design issues. These issues can be tackled upfront by identifying the performance bottlenecks right at the time of design by defining a systematic design strategy. Use of proven design patterns and best practices evolved through learning and sharing are the best ways to address these bottlenecks.

---


Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2004, TAC.
About Java Technologies Practice
TCS meets the burgeoning demand of internet based and web enabled applications the world over and has developed some of the largest business applications in the world using cutting-edge technologies such as Pure Java, J2EE, Portals, Content Management and Framework based solutions. The combined knowledge of business domains and technology puts TCS into unique position of delivering complete internet based and web enabled solutions to its customers. The Java Technologies Practice manages relationships with large IT vendors which helps us to gain early access and develop skills on the latest products and offerings in the internet related area. The practice has evolved due to its continuous tracking of developments in the area of java based technologies.

About Tata Consultancy Services
Tata Consultancy Services (TCS) is among the leading global information technology consulting, services and business process outsourcing organizations. Pioneer of the flexible global delivery model for IT services that enables organizations to operate more efficiently and produce more value, TCS focuses on delivering technology led business solutions to its international customers across varied industries.

For more information contact
Vinay Deshpande
Tata Consultancy Services Limited,
TCS Gateway Park,
Akruti Business Port,
Road-13, MIDC,
Andheri(E), Mumbai-400093
India

Phone: +91-22-5550 6775
Fax: +91-22-5660 6814

Email: vinay.deshpande@tcs.com
Website: www.tcs.com/java