WHITE PAPER

Presentation tier performance optimization

The performance of websites was always a critical non-functional requirement. A better performing site directly translates into better user experience, repeated site visits, and hence increased revenues. A fast performing site invariably provides a competitive edge as well. Also these sites are often indexed faster by search engines and often appear at the top of search results[8].

An ever increasing expectation for performance of websites means that Web pages need to be designed to be optimal and fast rendering. Many a times, websites lose their customers in a blink of an eye if they are a sub-second slower than their competitors[1]

This white paper discusses this critical non-functional requirement, methods to achieve it, and draws examples from various real-world scenarios. Most of modern applications follow layered architecture mainly consisting of three key Layers: presentation layer, business layer and integration layer. An optimization approach invariably involves all the layers. This white paper mainly focuses on the optimizations in the presentation layer.

Infosys
be more

Need for Speed

Not long back even a 2-second page response time was considered as an acceptable one. However, web users have become increasingly impatient when it comes to speed these days. Earlier, speed was considered a feature and now it is deemed a necessity. Additionally, technological innovation in mobile space has raised the bar for speed. Hence, speed makes a lot of economic sense now.

A recent research found that 250-450 milliseconds are the magical numbers that decide the winner in the race of web speed[1]. Research also indicates that the slower the site, the lesser would be the number of clicks and transactions performed on the site, which would eventually result in the loss of users[2].

Strategies of presentation layer performance optimization

There are broadly two main categories adopted for performance optimization:

a. Bottom-up strategy: This involves carefully laying out the ground rules for performance based on required SLAs and design / develop the pages by adopting the principles laid out. This is the preferred approach that involves optimization in both presentation layer design and development.

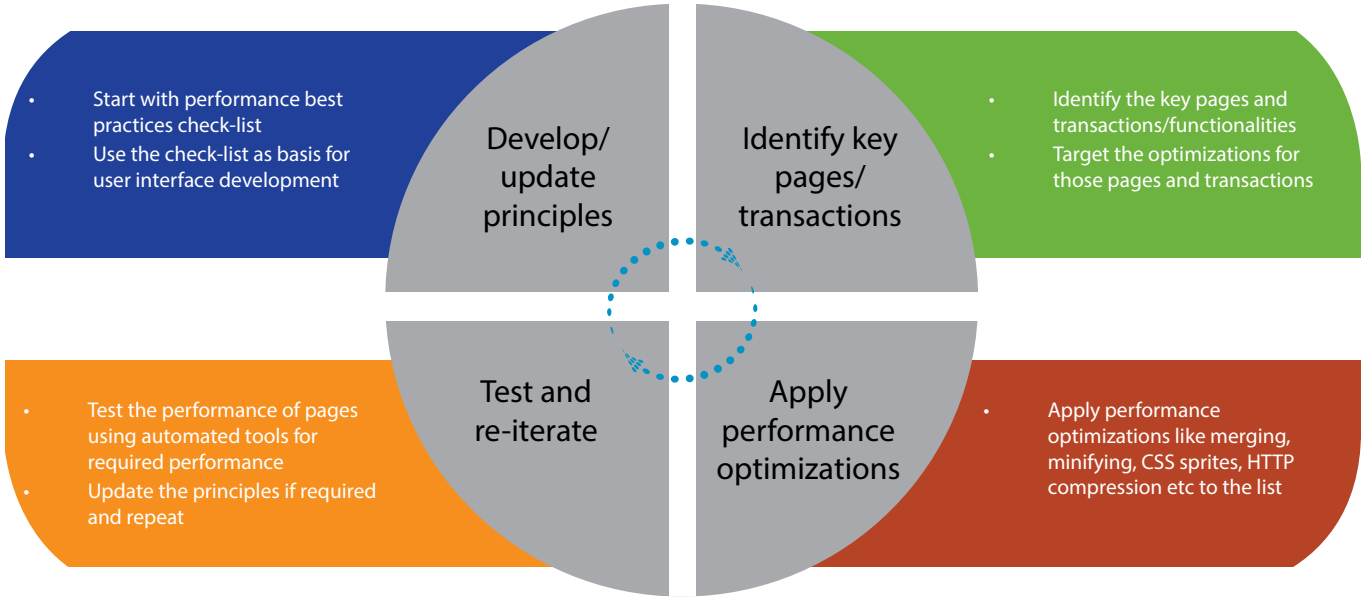
b. Top-down strategy: This is a reactive strategy which involves doing a post-mortem of pages when any performance issue is discovered. This involves analysing the page components and targeting the optimizations which reap big and quick benefits and iteratively enhance other components. This is potentially costly and the cost mainly depends on the phase during which the issue is uncovered.

Subsequent sections discuss these strategies in detail.

Bottom-up performance optimization

This strategy essentially is performance-by-design wherein performance optimization principles are framed, applied, and maintained right from application design phase. This is the preferred strategy to incorporate performance as a core development principle instead of having it as an afterthought which involves following key stages:

1. Layout performance principles
2. Execute the performance principles by identifying the key pages/transactions and optimizing them
3. Monitor and maintain the performance SLAs



Step – 1: Layout performance principles

An organization can create a performance guideline framework that forms the guiding principles for individual projects. The performance guideline framework can be constructed based on:

The performance service level agreement (SLA) the organization has promised to its customers :	i. Page render time across different geographies ii. Overall transaction time for key processes.
Making the user interface rich as well as responsive:	i. Richness involves providing feature rich client-side components like widgets and pages with high degree of intuitiveness ii. Responsiveness and interactivity involves making those client-side components highly interactive by adopting techniques like partial-page-rendering.
Request optimization for a page which mainly involves:	i. Minimize number of HTTP requests ii. Minimize amount of data requested per request by maximizing usage of client-side components wherever possible and employ partial page rendering to avoid the full page refresh.
Adopting wide variety of caching strategies across various layers which involve things like	i. Using cache headers ii. Caching lookup lists iii. Caching can be applied at webserver, server side, database layer and other possible integration layers iv. Avoid 3rd party plugins unless absolutely required. Even when they are included, only load the scripts on-demand and keep the 3rd party scripts at the bottom of the page.
Optimizing static assets like images, JavaScript, CSS, JSON which involves	i. Minimizing number of JS/CSS/Image files by minifying and merging them ii. Appropriate positioning of the files to improve the perceived page load time for user. iii. Use PNG format of image always iv. Encouraging usage of CSS sprites v. Use lazy loading of content wherever possible vi. Avoid iframes and redirects to the best extent possible
Heighten the user experience of key business processes:	i. Understand the key application expectation from end uses by conducting usability studies, surveys, interviews and A/B testing. Once the key expectations are understood design performance strategies around these expectations ii. Adopt process improvements such as: 1. Reduce number of steps/pages for completing the overall process 2. Automate the steps in process to the extent possible. 3. Provide quicker alternatives for a business process like one-click checkout, guest shopping etc.

Step – 2 Execute performance principles

Once the performance ground rules are set, individual project teams can implement the performance guidelines by adopting various optimization techniques applicable to their context. Optimization occurs at two levels:

- During application design
- During application development

Application Design Optimizations

Some of the design optimizations emerge from performance guidelines laid out in step – 1. Following list provides few key design considerations from performance point-of-view:

Simple and Lightweight: Keep the frequently used pages like home page and landing page simple and yet effective. This would involve things like	i. Including only key functionalities to keep it light weight ii. Having optimized marquee images
Provide interactivity through usage of client-side components:	i. Wherever possible, employ partial page rendering to avoid the full page refresh. ii. Adopt client-side validation framework to catch the errors
Provide intuitive information architecture and easy navigation after careful and thorough research of key functionalities. This involves things like:	i. Providing optimized search functionality on all pages to reach any page using keyword search ii. Providing elaborate menu to allow the user to navigate to any sub level page.
Provide multi-channel interface:	i. Provide alternate versions of pages / functionalities to multiple devices ii. Provide cross-channel integration for seamless user experience
Enhance user overall experience by leveraging client-side components:	i. Provide personalized experience based on user's previous browsing history. ii. Remember user's preferences to customize the key functionalities such as search, navigation iii. Recommendations based on transaction history iv. Leverage web analytics to gain insights into customer's implicit preferences and use it for further customizations
On-demand data loading:	i. Wherever possible load the data only on-demand instead of pre-loading all the data. For instance: 1. Use pagination for search results and load the second page only when requested 2. Information in the popup can be loaded only when user clicks on popup link.
Identify areas for automation:	i. Come up with process for automating and make it part of process. For instance merging and minification can be automated and added to the build process ii. Identify all the tools which help in automation and optimization to improve productivity 1. Use tools like Google PageSpeed, Yahoo YSlow etc. for constantly checking the page performance. 2. Use tools like Gomez for identifying page performance across geographies. 3. Identify Web analytics tools for automatically tracking the user activities and page performance.
Avoid chatty service calls	i. Minimize the service calls from presentation tier
Encourage asynchronous calls	i. Wherever possible promote asynchronous calls between presentation tier and business tier.

Application development performance optimizations

Before discussing the techniques for performance optimizations, let's look at the factors which contribute to page performance

Normally performance optimization is a multi-layer (presentation layer, business layer, database layer etc.) and multi-team (UI developers, business logic developers, networking team, and Infrastructure team) effort. This paper primarily addresses the concerns related to the presentation layer.

Factors affecting page performance

Today's websites are expected to be highly user intuitive and feature-rich. They almost always compete in the online Olympics to serve "richer, friendlier and faster" content. During development of some of these features, developers would inadvertently compromise performance. Let's examine few of the key features and their impact to page performance in detail:

Feature	Factors affecting page performance
Rich and Intuitive User interface using client side modules	• Inclusion of multiple JavaScript libraries which build the UI modules and impart interactive behaviour. This increases resource requests for a page. • Inclusion of multiple CSS files for adding styles for the UI modules. This increases resource requests for a page. • Including the JS files at the top of the page which blocks the loading of subsequent page components.
Support huge number of functionalities on the home and landing pages	• More the functionalities more will be the Document Object Model (DOM)/page size. • Possibility of duplicate resource requests. • Possibility of internal or external network bottlenecks.
Integration with 3rd party functionalities	• Include ads before the page load • Include lots of plugins like Twitter, Facebook, Google+, and LinkedIn etc. • Include JS files required for site survey at the beginning of the page
Adding large marquee images / animated flash objects/videos on each page	• Multiple images increases the resource requests • Bigger the asset size, larger would be the bandwidth consumption
Security requirements preventing usage of client-side components	• Absence of client-side page components forces the user to load the full page for each functionality. This would increase the overall time taken for completing a transaction. • Absence of server side caching would aggregate the performance issue
Enable mobile access	• Absence of device specific site and mobile accelerator would render the page slow in mobile devices

A deep-dive into User interface performance optimization techniques

Following table provides a comprehensive list of performance optimizations that can be carried out at a user interface level:

Performance Optimizations table

UI performance optimization	Performance gains	Tools and Techniques
Merge all the JS and CSS files	• Minimize the total number of resource requests. • Network round trip for additional resource trip is avoided	• YUI
Minify: • Minify the JS and CSS files • Minify the HTML content	• Reduced size would minimize the response time • On an average there will be 30% reduction in the file size for JS and CSS files. • On an average there will be about 20% size reduction in page size.	• JSMin • YUI
Asset Placement: • Place all external JavaScript links at the bottom • Place all CSS file links in the head element	• As JS files are at the bottom of the page, it will not block the serial load of the page. This would decrease the perceived load time for the user. Many times 3rd scripts related to analytics and others has the potential to slow down the entire site. Hence placing them at the bottom would eliminate this problem	NA
Assets Optimization • Use CSS sprites • Use lossless compression or PNG files	• Results in multi-fold benefits such as reducing the number or resource requests and reduced image size • Image compression would decrease the size by approximately 25%	• Smushit Image compressor • Trimage
Compression • Enable gzip compression for HTTP traffic	• Absence of device specific site and mobile accelerator would render the page slow in mobile devices	• Content-Encoding header
Choose appropriate presentation components	• Use the JavaScript library which is enough to satisfy application requirements. • Avoid loading unnecessary components of the JavaScript library which are not used for page. For instance while using DOJO library, we can create custom build to load only the required components.	

Caching

Caching requires a special mention among optimization techniques as it has potential to drastically impact the page performance if optimally employed. Following are some of the caching techniques that can be employed at presentation tier:

Browser caching

- Leverage browser caching by leveraging headers like "cache-control" and "expiry" for static assets.

Web server caching

- Store the static assets in web server which are fine-tuned for delivery of these assets
- Leverage the cache settings for rendering the static assets

Cache controlled values on client side

Normally Web applications display list of controlled values. For instance, a typical registration form displays the list of allowed countries in a drop-down. In such cases instead of reading the complete list of controlled values from the data source every time, it can be added to a JSON file and cached as a static asset. The JSON can be refreshed based on update to the source data. Similarly the client-side validation framework requires things like data patterns, validation business rules which can be stored in client-side components like JSON/XML file.

CDN caching

Sometimes clients will be having performance specific requirement for each geography. Let's say that we would like to render a normal page across all geographies when the origin server is in the US, then even with fully cached and an optimized site, it would be challenging to render the page within 2 seconds in the Asia Pacific region.

In such scenarios it is worth considering the Content Delivery Network (CDN)/Edge caching like Akamai, Amazon CloudFront. CDN systems would cache assets like images/Videos/JS/CSS and would serve the content from geographically optimal location to accelerate the page rendering.

Step – 3 Monitor and maintain performance SLAs

Post-production the application needs to be continuously monitored for performance issues. Following are some of the methods for implementing the same:

Monitoring

Following are some techniques that can be used for post-production page monitoring.

Page performance monitoring

Few web analytics tools like Omniture provide the page load times which can be constantly monitored for any performance issues

Real time cross geography monitoring

Systems like Gomez networks provides real time monitoring capability which monitors the site from various geographies on frequent basis and notifies the site admin if the site performance falls below specified threshold.

Additionally in-house custom tools can also be developed to do a health check of servers and page load. A combination of effective in-house and external monitoring infrastructure helps the site admins to better respond to performance situations.

Application Monitoring

Post production release the application needs to be constantly monitored to identify any performance bottlenecks and performance issues. Tools like IBM Tivoli monitoring and IBM Tivoli composite application manager (ITCAM) can be used for this purpose

Maintenance

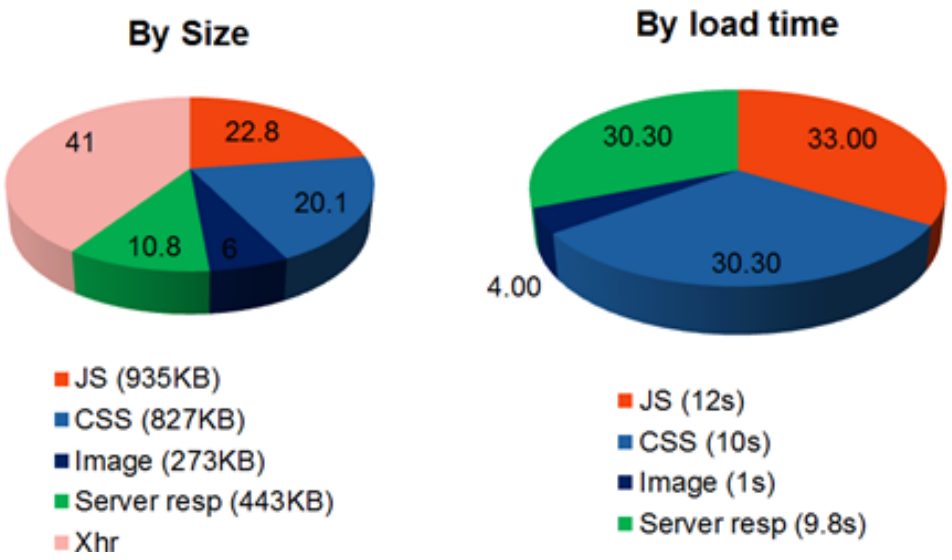
- All production updates should strictly adhere to the performance guidelines outlined in step above. For instance if there is a need to add additional JavaScript page, it must be minimized and merged with the existing master script to minimize HTTP request
- While adding additional functionalities, client-side version of that functionality needs to be given preference to avoid full page refresh.

Case studies

Following are some of the real-world case studies related to performance optimization

Case study – 1: Banking client's site performance optimization

Following graphs indicate the asset (JS, CSS etc.) contribution to overall page size and page load time:



1. Context and Analysis

- The dashboard page consisted of large number of functionalities.
- To support large number of functionalities it also needed good number of scripts and CSS files
- The page was taking about 30 seconds for the first time load and about 10 seconds for subsequent loads

2. Root cause

- The performance optimization was an after-thought rather than a bottoms-up approach.
- The creative agency had missed the performance optimizations for the delivered web artifacts that includes HTML, JS, and CSS etc.
- The dash board page had too many portlets which required lot of server side processing

3. Performance Optimization Exercise

A thorough performance optimization exercise was carried out at all layers. Following were the recommendations made:

a. Business layer:

- Server side caching and connection pool was recommended.
- Batching of services call was recommended to reduce the round trips to services layer.

b. Presentation layer:

- All the performance optimizations recommended in the performance optimizations table (merging and minifcation, asset placement, browser caching etc.) was recommended.
- A CDN network like Akamai was recommended to maintain acceptable page load times across geographies.
- It was recommended to reduce the number of portlets on a dashboard page. The optimal size of portlets on a page is between five to eight.

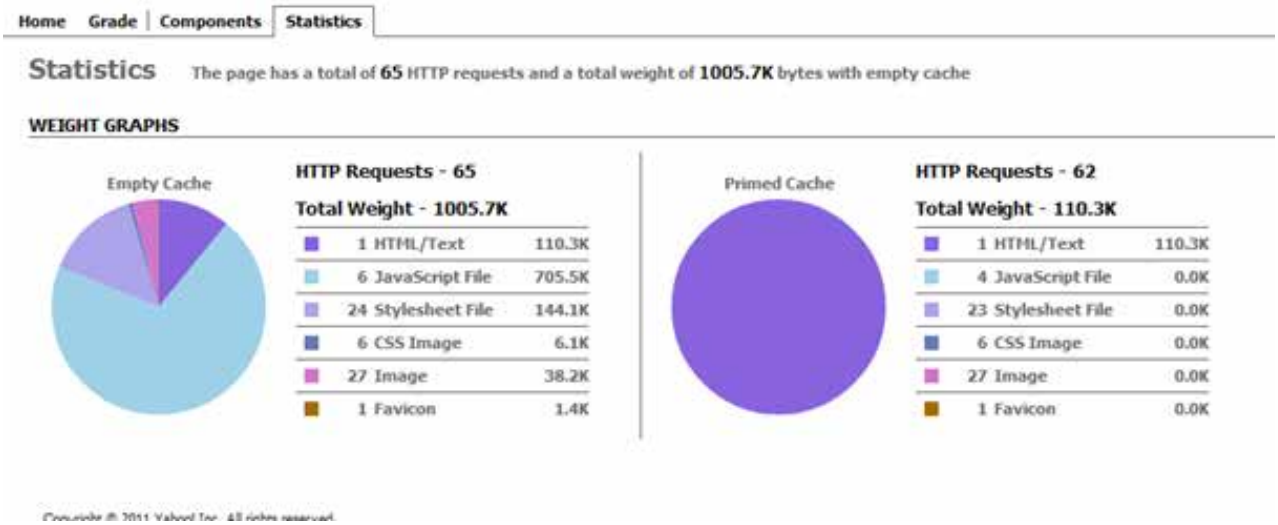
4. Results

a. Page load times improved to acceptable levels of 10 seconds

Figure 2: The proposed reporting mechanism

Case study – 2: Semiconductor based Manufacturing client's site performance optimization

Another example of a frequently visited page for a North American based semiconductor major is shown below:



1. Context and Analysis

- The home page of the manufacturing site consisted of multiple JS and CSS files
- The CSS and JS files were not minified (?)
- Most of the functionalities were doing a complete page refresh which made the user to wait.

2. Root cause

- Development team had missed out the presentation layer best practices.
- Partial page rendering was not adapted to the fullest extent.

3. Performance Optimization Exercise

Following were the recommendations made:

a. Business layer:

- Out-of-the-box server side caching was widely adopted for all content/data fetched from external systems

b. Presentation layer:

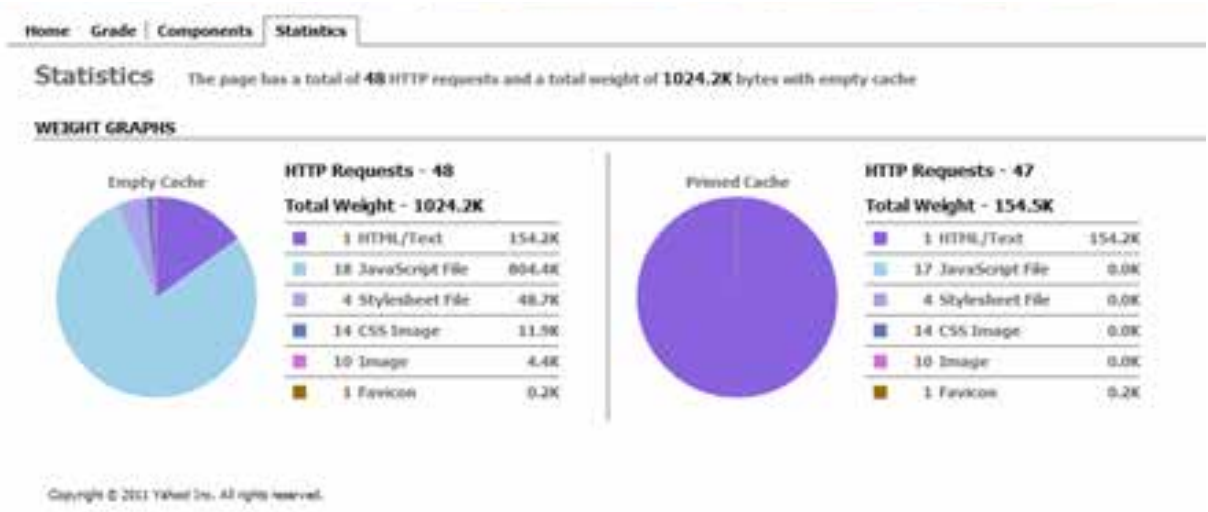
- All the performance optimizations recommended in the performance optimizations table (merging and minifcation, asset placement, browser caching etc.) was recommended.
- A CDN network like Akamai was recommended to maintain acceptable page load times across varied geographies.
- AJAX based client side modules were suggested for key functionalities like search, product finder to enhance user experience

4. Results

- As a result of these performance optimizations all HTTP based pages were rendered within 2 seconds across all geographies
- All HTTPS based pages were delivered within 5 seconds across all geographies.
- Page views increased by 100% to 80,000 as compared to previous year owing to user experience improvements caused by client- side modules.
- Overall customer satisfaction increased to 81%, a major chunk of contribution came from performance improvements in HTTPS pages and client side modules.

Case study – 3: Top engine manufacturing client's site performance optimization

Following graph indicates the asset size to page size ration of a supplier portal for energy major in North America:



1. Context and Analysis

- Supplier and distributor portals were taking about 25 seconds in the production environment.
- Much of the page load time was spent in server calls and loading heavy static assets.
- The database access layer consisting of Hibernate was not optimized.

2. Root cause

- Web components were not optimized using performance best practices.
- Server side code was not optimized

3. Performance Optimization Exercise

Following were the recommendations made:

a. Business layer:

- Batch the database calls and use Hibernate caching, lazy loading to enhance the database operations.
- Use on-demand pagination instead of pre-fetching a large result set.
- Rewrite the queries to use the database indexes for faster performance.
- Use optimal application server parameters related to thread and database connection pool.

b. Presentation layer:

- All the performance optimizations recommended in the performance optimizations table (merging and minification, asset placement, browser caching etc.) was recommended.
- A CDN network like Akamai was recommended to maintain acceptable page load times across geographies.
- AJAX based client side modules were suggested for key functionalities like pagination and dashboard functionalities.

4. Results

- Page load time reduced to within 10 seconds.

Top-down performance optimization

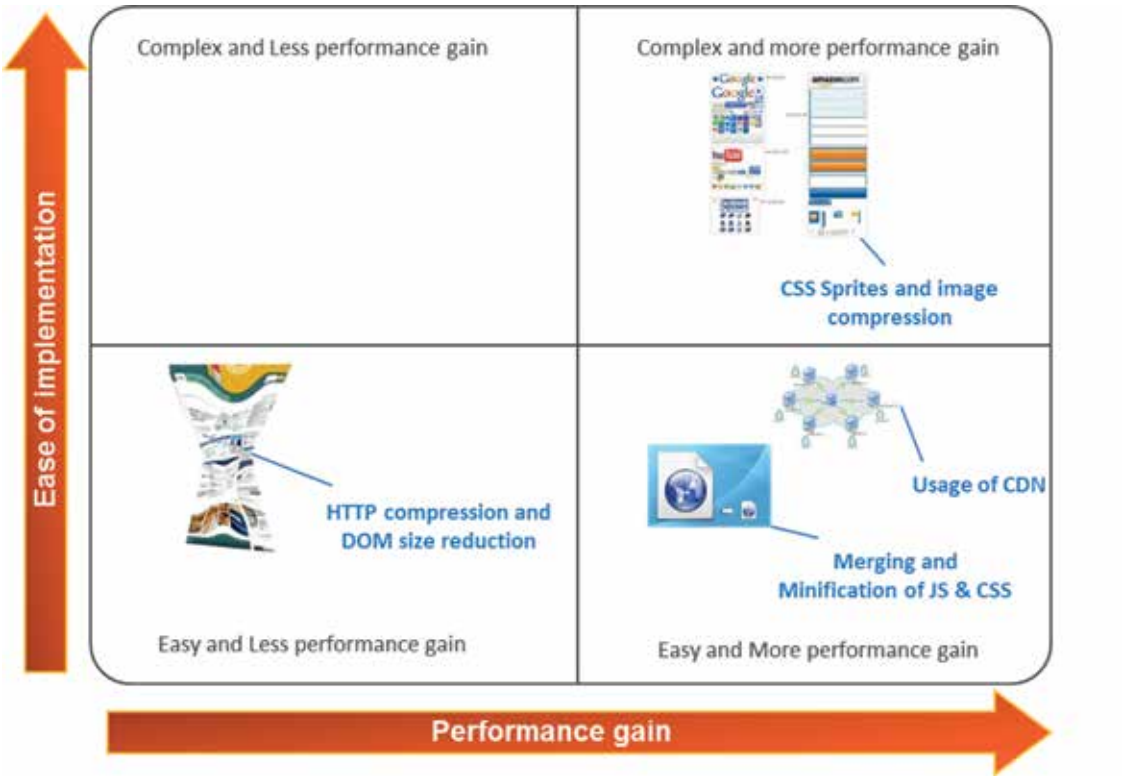
This strategy is employed when an existing application needs to be optimized for performance. This is preferred only when we are optimizing legacy applications; all newly developed applications should use bottom-up strategy as the cost involved in top-down optimization is high and optimization options are limited.

Getting the pages on a fast lane

Pareto's principle of 80-20 rule applies well to performance related issues. This is true for both analysing root cause and for fixing the issue.

One would start with the critical performance optimizations that would result in major benefits in the initial iteration and involve the remaining ones in subsequent iterations.

Following chart depicts the key performance optimizations and the performance gains that can be realized:



Following are the high level steps involved in top-down performance optimization:

1. Analyse the factors contributing to the page performance. We can use tools like PageSpeed, YSlow for performing this operation.
2. Start optimizing the activities which would realize maximum performance improvements. Above chart provides a guideline for this step
3. Iteratively implement other optimizations with subsequent releases.

Note:

1. The application design and the time available limits the performance optimization options in this strategy. For instance if the complete application is developed without any client-side components, then developing client-side widgets would potentially involve major application framework changes leading to higher development and testing times.
2. Application domain requirements need to be kept in mind while performing this step. For instance if security requirements don't permit usage of client-side components, then all optimizations related to that would not be applicable.

Few other dimensions

Real world scenarios are bit more challenging than what we have discussed till now. These would pose its own challenges for performance. Few of them are discussed below

Transport level security

Most of the financial sites and other sites which transmit confidential information employ transport level security and serve content over HTTPS. As the delivered content is encrypted at the transport layer, there is an obvious performance trade-off for this. Following are some of the options that are worth considering in this scenario:

1. Carefully evaluate if a page really needs HTTPS. If it just displays public content and campaigns, it is better-off to serve them over HTTP instead of HTTPS
2. CDN networks also have modules which would speed up the delivery for SSL sites. Consider employing them. For instance Akamai offers secure content delivery.

Additionally, we can also consider using hardware SSL accelerators like that of Blue Coat and Coyote Point.

Mobile device support

As the mobile internet traffic is increasing, following are few thoughts to keep the site optimized and ready for mobile devices:

1. Support an alternate version of your site for mobile devices which has an optimized layout and images to suit the mobile platforms.
2. Keep the pages light and load the JS code through XHR call
3. Many servers provide mobile accelerators

Security vs. Speed

While building financial sites, there could be instances wherein the performance principles cannot be fully applied due to security constraints. For example:

- The financial firm has restriction for using client-side modules and functionalities.
- Restriction for AJAX proxy which prevents content served from different domains
- Mandatory usage of HTTPS for all pages
- Heavily loaded pages to incorporate multiple functionalities

In such instances optimization can be done mainly for the server side components.

Future outlook

Some of the future trends in performance optimization are given below:

1. HTML 5 offers a whole array of features related to performance optimization. This includes hardware acceleration support for graphics and video
2. Offline web apps features in HTML 5 can be leveraged to serve static informational content.

About the Author

Shailesh Shivakumar

is a Technology Architect with the Manufacturing unit at Infosys. He has over 10 years of industry experience. His areas of expertise include Java Enterprise technologies, portal technologies, User interface components and performance optimization.

He can be reached at shailesh_shivakumar@Infosys.com