A Survey and Analysis of Techniques and Tools for Web Performance Optimization

Shailesh Shivakumar, P.V Suresh
Indira Gandhi National Open University
India
shailesh.shivakumar@gmail.com

ABSTRACT: *The performance of web applications is of paramount importance as it can impact end-user experience and the business revenue. Web Performance Optimization (WPO) deals with front-end performance engineering. Web performance would impact customer loyalty, SEO, web search ranking, SEO, site traffic, repeat visitors and overall online revenue. In this paper we have conducted the survey of state of the art tools, techniques, methodologies of various aspects of web performance optimization. We have identified key web performance patterns and proposed web performance optimization framework. We have elaborated on various techniques relate to WPO such as asset optimization, caching, pre-fetching, infrastructure optimization, performance patterns and anti-patterns, monitoring and testing. We have compiled the latest trends and potential research areas in WPO area.*

Organization of the paper: *The paper is organized as follows. We will start with introduction concepts of WPO and we will then examine various aspects of WPO such as caching, design checklist, asset optimization, pre-fetching, infrastructure optimization, server optimization, performance monitoring, and performance testing. In the final sections we will look at the emerging trends and scope for future research areas in WPO.*

Keywords: Software, Software Engineering, High Performance Computing, Software Performance

Received: 16 October 2017, Revised 5 December 2017, Accepted 17 December 2017

DOI: 10.6025/jio/2018/8/2/31-57

© 2018 DLINE. All Rights Reserved

1. Introduction

A faster web would have a positive impact on user experience and ultimately leads to increased user satisfaction and impacts the revenue [2]. Speed of web pages is also factored while ranking the web pages [1] and e-commerce sites show increased revenue [3] with improvement in page load time (PLT). User traffic and repeat visits is also related to the site performance [56] [57]. The survey [71] finds that site download rate (including the initial access speed, display rate, speed of navigation across pages) directly impacts the perceived success rate by end users.

WPO involves best practices and techniques to increase the speed of web pages [87]. WPO includes page components such as HTML content, presentation components, page elements, page assets and such. WPO provides techniques, best practices, thumb rules, methodologies for end-to-end web performance optimization.

1.1 Impact of WPO

WPO has impact in following aspects:

• **Customer Churn:** Research indicates that customers would abandon the slower web pages [88] [91] [92].

• **User Impact:** User experience is drastically impacted due to page performance.

• **Site Traffic:** Site traffic is impacted if the page takes more than 3 seconds [89] and most users expect the page to load within 2 seconds [89].

• **Revenue:** For e-commerce sites, the shopping and check out performance is crucial for the conversion.

• **Multi-device optimization:** An optimized web page also impacts the performance on various devices.

1.2 Web Request Processing Pipeline

To serve a HTTP web request, following times are involved:

1. DNS lookup time to look up for the domain server. This depends on the latency of the connection and usage of DNS cache.

2. TCP connection time to complete the 3-way handshake with server. This includes 3 packets for SYN, SYN/ACK and ACK between client and server. This too depends on the latency of the connection.

3. Server response time (or server think time) is the total time taken by the server to start sending the response. Server response time is indicated by Time to first byte (TTFB). The total time needed for the browser to see the first byte response since the request is also known as Time-to-first-byte (TTFB).

4. Object download time is the total time required to download the requested object over HTTP connection.

5. Page load time is the total time required to download all the objects in the web page.

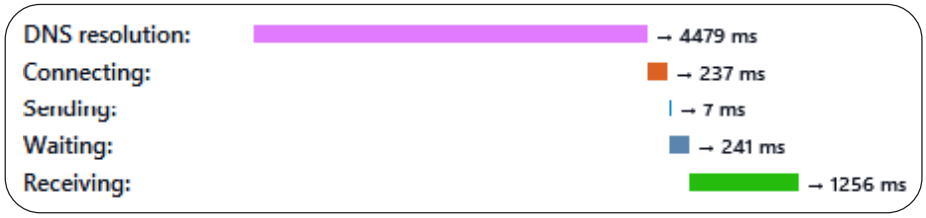

Figure 1. Request Processing Pipeline

In Figure 1 the DNS lookup time is 4.4 seconds and TCP connection time is 0.23 seconds and the Time to First Byte (TTFB) is 4.964 seconds (the sum total of DNS look up time, connection time, sending and waiting time). Once the content is received, page DOM is loaded and the content is rendered.

The network devices can cache DNS records [69] and TCP connections [68] [70] to speed up time needed for DNS lookup and TCP connection reducing the latency. The research [68] finds that connection caching and DNS caching would improve the load times by 35% and 10% respectively.

1.3 Web Performance bottlenecks and Web Performance anti-patterns

Let us look at common performance bottlenecks and anti-patterns which impact the web performance. Table 1 provides a list of commonly occurring performance bottleneck.

1.4 Dimensions of WPO

The process of Web optimization can be analyzed from several dimensions:

Bottleneck Area	Performance anti-patterns
Web page Design	• Bad design of key pages (such as gateway pages, home pages or landing pages) by including numerous images and presentation components. • Not conducting iterative performance testing to assess the page performance across geography. • Absence of real time performance monitoring and notification infrastructure • Absence of layer-wise caching strategy. • Usage of known performance blockers such as i frames. • Using uncompressed images and scripts on the pages. • Not doing a Omni-channel testing for all pages. • Not following performance design best practices. • Not adopting a sound multi- layer caching strategy.
Third-party components	• Third-party Scripts and widgets would block page load and impact overall page performance.
Network bandwidth	• Usage of sub-optimal network bandwidth across internal systems
Server configuration	• Not adopting optimal server settings for parameters such as heap memory, thread pool size, connection pool size etc.
Infrastructure capacity	• Usage of sub-optimal memory, CPU, disk capacity for servers. • Not conducting load testing, stress testing, endurance testing and related performance tests.
Performance testing	• Not conducting all necessary performance tests (such as load test, stress test, endurance test) for web application. • Conducting performance testing at the end of the application development. • Not conducting Omni- channel testing to test performance on mobile platforms.
Page code	• Not conducting performance code review. • Not performing iterative performance testing.
Service calls	• Having numerous and chatty service calls on crucial pages slowing down the web page. • Using blocked and synchronous service calls would impact page performance.

Table 1. Performance Bottleneck & anti-patterns

• **Optimization of Request Pipeline Processing Systems:** In this category we will examine all the systems involved in the web request processing pipeline. This involves browser software, CDN, proxy server, network, load balancer, web server, application server, integration middleware, backend services, database server and such.

• **Client-side and Server Side Optimization:** Client-side performance optimization includes all optimizations performed on client-side presentation components such as HTML pages, images, assets and such. Server side optimization includes performance tuning of server –side components such as fine-tuning business components, setting optimal server configuration, right infrastructure sizing and such.

• **Design Time and Run Time Optimization:** Design time optimization includes the static and offline performance optimizations activities such as performance code reviews, performance testing, and offline performance tuning and such. Run time and dynamic performance optimization activities include real-time performance monitoring and notification, run time performance optimization and such.

• **Web Component Optimization:** Another aspect of web optimization is to optimize each of the constituent's web components such as HTML, images, JavaScript, CSS, Rich media files and such.

We will be covering various aspects in all these dimensions throughout this paper.

2. Web Performance Patterns and Web Performance Optimization Framework

2.1 Web Performance Patterns

Given below are some of the key web performance patterns for optimal performance:

• **Minimal Round Trips:** The web page should minimize the server calls to the extent possible. Wherever possible the calls should be batched to minimize the calls.

• **Asynchronous Loading Pattern:** All the page assets should be loaded asynchronously.

• **Lazy Loading Pattern:** The page assets should be loaded when required and on-demand.

• **Light Weight Design:** The page should be designed with light weight principle.

• **Device Specific Rendition:** The page content, assets should be optimized for the rendition device.

• **Responsive Page Content:** Responsive design for HTML elements and adaptive design for content should be followed.

2.2 Web Performance Optimization End-to-end Flow

Various layers and systems involved in a typical web request processing is shown in figure 2.

Figure 2 identifies various performance optimization measures that can be taken at each layer. Web request originates from user agents (user as browsers) or from user devices (such as mobile devices, tablets and such). In this layer the browser cache is often used for page performance optimization. The request then goes to content delivery network (CDN) which mainly caches static assets such as images, videos, static pages and such. The web servers and proxy servers mainly cache static page assets. At web server layer we can set the caching and cache header rules. Asynchronous asset loading and responsive web page design are other performance optimization techniques which can adopted in this layer. At the application server layer we can leverage server-side caching frameworks, distributed caching model, parallel computing and use continuous and iterative performance testing. At the backend layer, we can leverage services cache to cache service responses, query result cache for caching query results, content cache to cache content and fragments and adaptive content design.

A properly sized infrastructure setup is needed for all the servers to ensure that all the systems in the web request processing pipeline properly scales for the user load. Monitoring infrastructure would constantly monitor the hardware components and provide report dashboards and real-time notification.

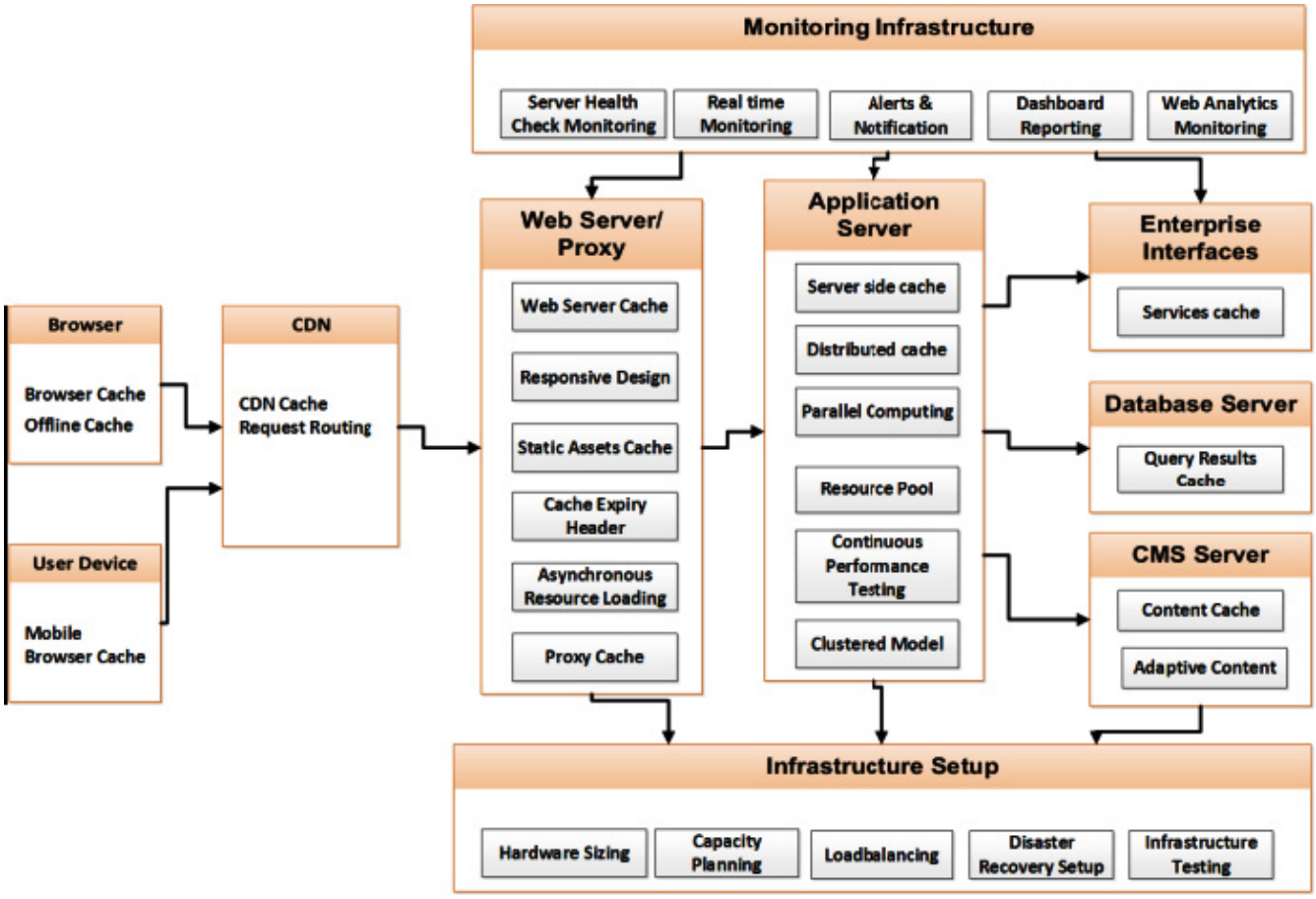

Figure 2. End-to-end performance processing pipeline

The primary infrastructure component used for caching is proxy server. Proxy server's cache the resources (such as page content, assets, documents, data and content) fetched from source systems and it would greatly improve the overall response performance. As far as infrastructure is concerned, the caching can be implemented at browser level, proxy level and at the origin server level [26] [27].

2.3 Web Performance Optimization Framework

Performance optimization Framework is depicted in Figure 3. The framework details various components and activities in four stages of web development.

During design and architecture phase, we could adopt performance based design. This includes defining performance design principles and developing a performance checklist and performance patterns which could be used during development and testing phases. We will also finalize the performance SLAs related to response time, throughput, and resource utilization and such. We would design performance test cases and setup optimal sized infrastructure.

During development, we would conduct performance code reviews.

Iteratively to uncover known performance issues. Multi-layer caching system would be developed. Asset and content optimization techniques would be implemented for page modules. Performance testing phase involves iterative performance testing and measuring all the identified performance metrics and SLAs. We would conduct various types of performance testing such as load testing, stress testing, endurance testing and volume testing. During performance testing and analysis, we would identify performance bottlenecks and fine-tune components and systems to address the bottleneck. Performance modeling activity involves identifying scenarios and performance objectives and metrics.

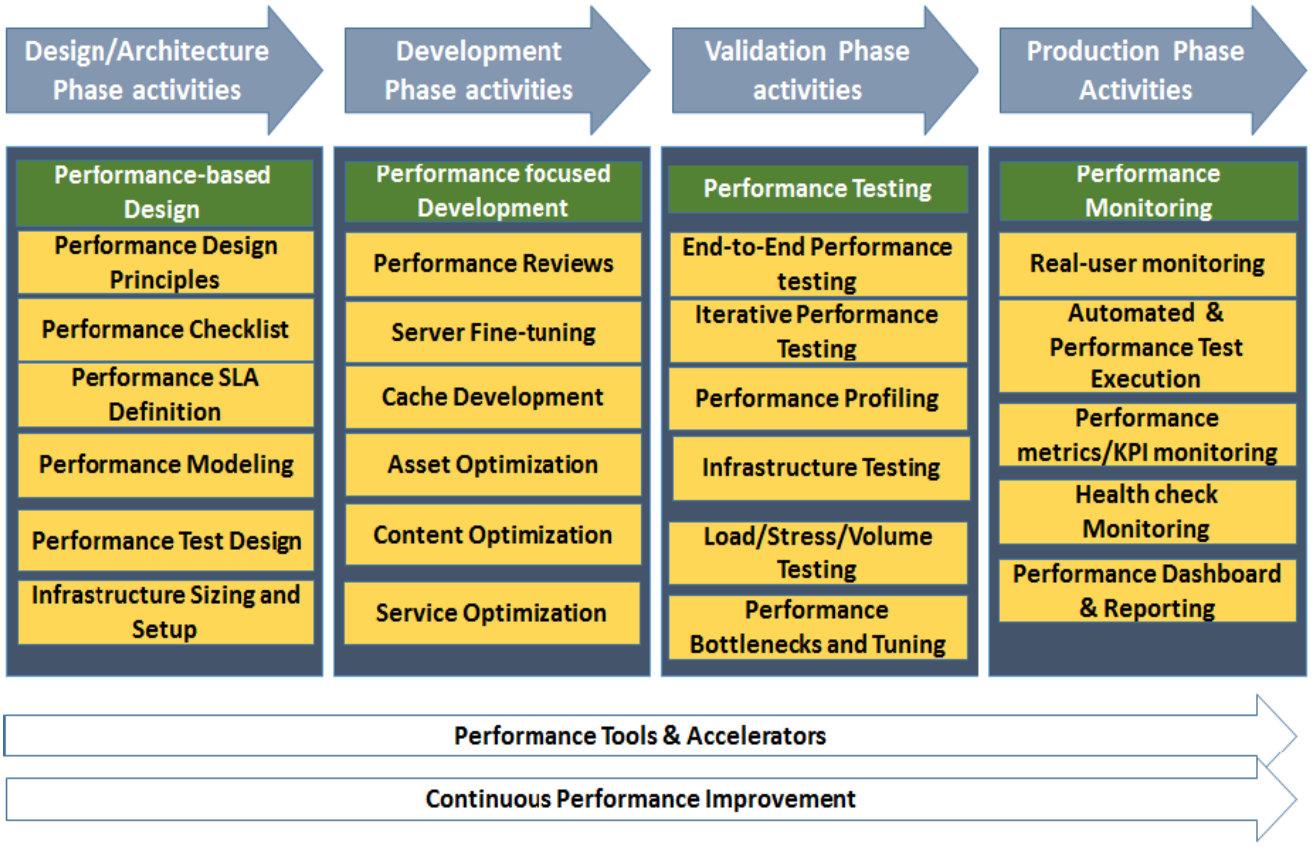

Figure 3. Web Performance Optimization Framework

In post-production phase, we would mainly conduct performance monitoring activities such as server health check monitoring, real-time application monitoring and such. SLA violations will be reported and notified to system administrators to take corrective actions.

3. Brief Summary of Techniques of Web Performance Optimization

In this section we briefly discuss the performance optimization techniques for web scenarios.

3.1 Web Performance Rules

Various best practices and thumb rules [85] [93] are elaborated for public web optimization. We will be elaborating on those well-known and proven web performance optimization rules in this section.

Once we get the list of all pages and processes we should adopt the following design-time best practices:

• Keep the key pages simple in design. This involves using only necessary UI components.

• Minimize the number of HTTP calls for the page. HTTP calls are required to load the resources and hence reducing the resources would automatically reduce the HTTP requests. It is recommended to keep the synchronous resource requests lesser than 20 requests per page.

• Explore means to load the page content asynchronously. We can leverage AJAX requests to load the page sections which provide non-blocking page loads.

• Ensure that there are no white spaces in the page to reduce the page size.

• Avoid using inline images, JavaScript and style sheets.

• Externalize all the JS, CSS and images.

• Use responsive web design (RWD) technique to cater to multiple devices and form factors. RWD consists of fluid grids, media queries that can auto-adjust based on the target device specifications.

• Ensure the page data is loaded only on demand and in lazy mode. For instance the list data or results data can be shown in paginated view and can be loaded only on user navigation.

• Maintain a performance checklist applicable for the technology platform and use it while developing the page.

• Keep the assets in optimized format. As this topic needs elaboration we have another section dedicated for this.

• Ensure that the page does not has any broken URLs.

• Ensure that page request does not redirect which results in additional HTTP request,

• Parallelize downloads across various domain names.

• Minimize page size. Preferably the overall page size should be between 100KB and 400KB for home pages and landing pages.

• Minimize session size and cookie size.

• Use asynchronous scripts and AJAX get requests.

• Ensure that all duplicate links, scripts, images, content are eliminated on the page. This would reduce unnecessary page requests.

• Perform iterative and phase-wise performance code review and leverage static code analyzers to pro-actively identify any performance issues.

• Perform server side optimization such as server side caching, distributed caching, optimal pool size and such.

• Perform infrastructure optimizations such as right server sizing, infrastructure testing, load balancing and such.

• Test the pages for all loads. Test the pages for all languages and geographies by simulating the requests.

• Test the page for all supported browsers and mobile devices.

• Business-critical processes should be optimized. This includes business process optimization, page design optimization, search optimization, check out/shopping process optimization, user registration optimization and such.

3.2 Content Optimization

A web page consists of multiple content sections. The content would come from HTML content fragment or from web content stored in CMS. Let us look at ways to optimize the web content retrieved from CMS.

• While designing the content strategy think of content in chunks instead of a monolithic content. Modular content chunk will make the content reusable and enhance the caching optimization.

• Use adaptive technique techniques (such as progressive enhancement/degradation) while creating the content. This will automatically make the content optimal for all devices.

• Cache the content at chunk level. Fine tune the caching period based on the content update frequency. Perform on-demand

chunk cache invalidation when new content is published.

• Adopt user-friendly information architecture for easier and faster discovery of relevant content

• Tag the content chunks with relevant metadata and tags that helps in accurate information discovery.

3.3 Security & WPO

Security is one of the key concerns for web applications. Security for web applications can be enforced at various levels. One of the most commonly used security constraint is to use secured socket layer (SSL) to ensure transport level security. SSL is a default choice for web pages hosting confidential information such as user credentials, user personal information and such. SSL would also impact the page performance [78] due to additional overhead. The most commonly used techniques for optimizing performance in such scenarios are as follows:

• Set proper expiration times for the objects so that browser can cache the objects appropriately [74].

• Use CDN which support SSL acceleration modules for forward caching.

4. Web Performance Design Checklist

In this section, we will look at the design guidelines and best practices for achieving optimal web performance. The books [66] [67] provide excellent performance guidelines from web performance stand point. Web developers and architects can use this as reference while developing web applications. Some of these optimizations are also available as filters for Apache's mode_page speed module [52].

As 80% [66] of load time is spent in making HTTP requests for non-HTML content, we could look at ways to optimize these web components in table 2.

Category	Performance Rule	Impact on Web Performance
Request Optimization	Reduce the number of HTTP Requests	Reduces the consumed bandwidth and data transferred o
	Merge the static assets such as JS and CSS files	Merging would reduce the number of HTTP requests and would improve the page response times by about 38% [66]
	Remove all duplicate file includes Remove all invalid URLs which result in HTTP 404	Avoid unnecessary and valid HTTP requests
	Load the JavaScript asynchronously	This would reduce the blocked loading of JS files
	Minimize usage of iframes	iFrames block the loading of parent window till iframe source is loaded and hence affects load times
	Minimize redirects	Minimize additional requests
	Cache DNS records	We can reduce the DNS lookup time through DNS cache maintained at browser level
	Remove any unused CSS, JS file includes	Minimizes HTTP requests

Web object size optimization	Minify JS and CSS files	Minification would reduce the size of JS and CSS file
	Compress images	Compressed images would reduce overall page size
	Leverage gzip for HTTP compression	Compression would reduce the overall page size by about 70% [66]
	Remove white space in the HTML document	Removal of white space would reduce the overall page size
HTTP Header Optimization	Leverage cache headers for static assets (images, JS, CSS, JSON and other binary files) using Cache-Control header with max-age directive	Allows browsers to optimally cache the assets
	Use expires header for	Avoids additional resource request
Asset placement	Place CSS files at the top	CSS elements in the head tag
	Place JS files at the bottom	JS files at the bottom would improve the perceived page load time. I would avoid the blocked loading of other assets
	Externalize inline JS or CSS	Enables browser caching and parallel downloads
Image optimization	Asynchronous image load	Load the images on-demand and in asynchronous when they are visible in the user's view port.
	Optimize image size	Use the right size image based on the requesting device
	Convert JPEG image formats to progressive format	This would reduce the overall image size
	Optimize image	Specify the exact width dimensions and height for all images
	Use image maps	Reduces multiple image requests
	Use CSS sprites Other techniques: inline images	All images are combined into a single one and the required image is displayed using style rules. This reduces number of image requests
Network optimization	Usage of CDN	CDN would optimize the resource request by serving the resource from nearest location to the requestor
	Use multiple asset hosting servers (for hosting images, videos and other multi-media content)	Allows browsers to download the content in parallel.

External Dependency optimization	Identify all external scripts and HTTP requests which impact the page performance and which block the page load and optimize them	
Web Application design optimization	• Perform regular and iterative performance testing to identify performance bottlenecks and fix them. Use automated and manual performance code reviews at regular intervals.	Iterative performance testing uncovers performance bottleneck during early stages
	• Use light-weight service based integration model and load the data asynchronously on-demand	

Table 2. Categorized Performance Rules

4.1 Tools for Web Performance Optimization

Table 3 below provides a list of tools that can be used for web performance optimization

Optimization	Open Source/Commercial Tool(s)	Comments
Category Web Page Analysis tools	• Yahoo Y Slow • Google PageSpeed • HTTP Watch • Dynatrace AJAX Edition	The tools analyze the page HTML and provide the performance benchmarking and improvement guidelines
Page development tools	• Firebug • Chrome Developer toolbar • Fiddler	Web developers can use these tools to analyze the page load times, asset size, asset load times and such.
Asset merging and minification tools	• Yahoo UI (YUI) minifier • JSMin	These tools can be used to merge JS/CSS files and minify them
Page Performance Testing tools	• JMeter • LoadUI • Grinder	The tools can be used to simulate the load on system
Image compression tools	• PNGCrush	These tools can be used to compress images
Web Server Plugins	• Mod_pageSpeed for Apache web server	The plugin provides filters for automatically implementing performance best practices such as compression, minification, merging, placement etc.
Web site Performance Testing	• https://www.webpagetest.org/	Test and analyze the performance issues for a specific web site
Real-time Application monitoring CDN tool	• Gomez (Commercial) • Akamai (Commercial)	CDN systems can be used to accelerate the content delivery across geographically distributed applications
Web Analytics	• Google Web Analytics • Omniture	Web analytics would give basic performance statistics about the web page and can be used to track user behavior and other metrics such as site traffic etc.

Table 3. Web Performance Optimization Tools

5. Asset Optimization

Static assets such as images, multi-media files and other binary files contribute to the web performance. On an average 70-80% of page contains non-HTML content such as images, JS, CSS [58] [66]. Hence optimizing the digital assets would be crucial for optimizing the performance of the overall web page. The books [66] [67] provides following techniques for optimizing the asset performance.

Page assets apart from the page HTML include the image graphics, media files (videos, flash), JavaScript files, Style sheet files (CSS files) and JSON based data.

Let us look at optimizing each of these assets in table 4:

Asset Category	Optimization Techniques
Images	• Use right format image for the appropriate channel. • Use CSS sprites to minimize image load requests • Compress images for optimal image sizing • We can resize the image with minimal size in PNG format • Use adaptive images to cater to multiple devices
Resource requests	Minimize resource requests by merging CSS/JS files to reduce multiple file requests
JS/CSS Optimization	All the JavaScript and Style Sheets should be merged and minified. This would not only reduce the HTTP requests but also reduce the impact on overall page size. Also we can place the merged and minified file at the bottom of the page to optimize the perceived page load times
Asynchronous asset loading	Wherever possible, load the assets such as images asynchronously and only on demand. For instance we need not load the image that is outside of the current view until the user does a page scroll.
Web-friendly data format	Use optimal data container such as JSON instead of using XML. JSON is optimized for web and is the lightweight alternative to XML.

Table 4. Asset Optimization Techniques

6. Caching

Caching is a key enabler for improving the performance of web. Caching has multi-fold impact on web performance. Essentially caching would fetch the required pre-processed from the nearest location, making easy and faster access to the required data. Caching would improve the page load times, service completion times, and perceived page response time and reduce the latency, reduces bandwidth usage and minimize the load on the source systems to improve its scalability, performance and availability. Besides aforesaid advantages, caching would also enhance the system scalability and uses network bandwidth optimally. Caching can be done at multiple levels: application level, Infrastructure level (server level, network level), Integration level (services and resource invocation).

6.1 Cache Architecture

The study [4] identifies three caching architectures: hierarchical caching architecture with various caching levels, distributed caching architecture with distributed cooperating caches and hybrid caching architecture consisting of cooperating caches at various caching hierarchy levels.

A detailed caching survey [4] discusses the essential properties of a web caching system such as fast access, robustness, scalability, stability, transparency, efficiency and simplicity. The survey categorizes caching systems into harvest cache [5] (consisting of hierarchy of mutually cooperating cache servers communicating through Internet Cache Protocol), cache mesh cache [6] (a distributed cache system which leverages smart cooperative cache placement and optimal cache routing), summary cache [7] (wherein proxy cache uses the summary of cache of other cache servers), adaptive web cache [8] [41] (a group of self-organizing, overlapping cache groups adaptable to changing conditions) and access driven cache [9] (uses proxy profiles and information groups to reduce the inter-proxy messages). The survey detailed in [84] provides caching techniques for dynamic content generation.

Based on the cached data type and nature of caching, we can broadly classify caching into two categories: static cache and dynamic cache. Figure 4 provides various sub-categories of static caching and dynamic caching.

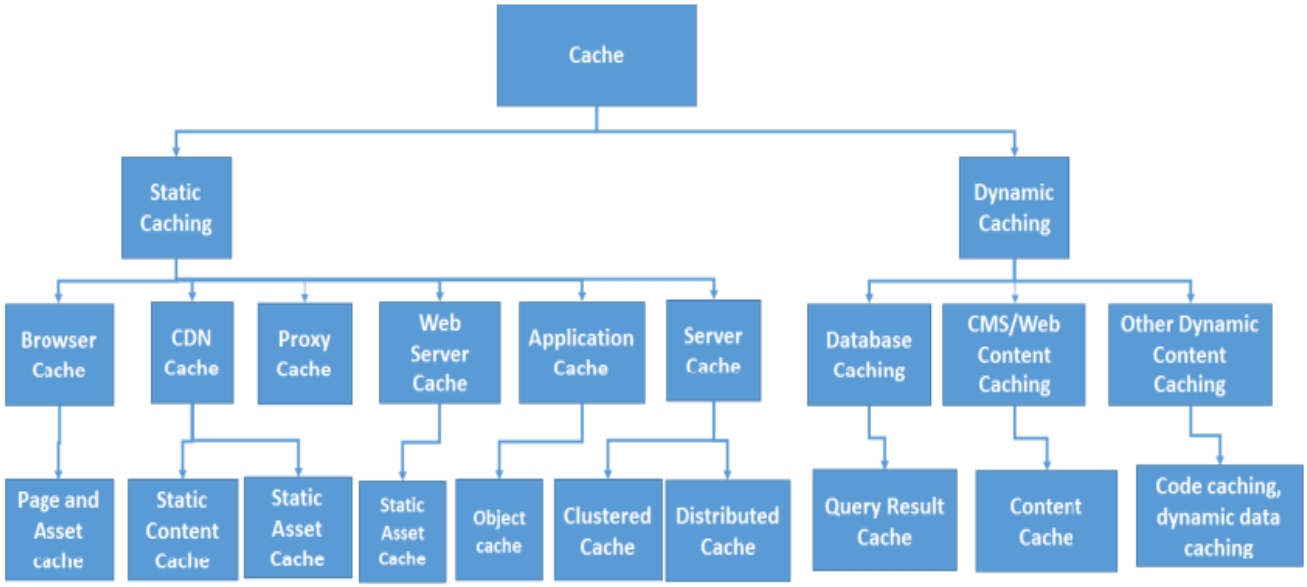

Figure 4. Cache Categories

In coming sections we will discuss more about static and dynamic caching.

6.2 Static Caching

Caching is a pre-dominant factor that influences the performance. The nature of the data cached in this category is static across users and devices. The data and content cached is globally used by all user sessions, pages and on all user devices. As the static cache content can be used globally, it can be pre-fetched and the cache can be "prepared" before the arrival for the request.

A multi-layer static caching at all layers should be adopted to ensure optimal performance.

6.2.1 Browser Level Cache

Browser cache is the nearest cache location for the end user. Browsers use HTTP headers to decide the caching details.

6.2.2 CDN Level Cache

CDN can mostly cache static global assets such as images, videos, static web content, static pages, multi-media files, JS/CSS files and such content.

6.2.3 Proxy Level Cache

Web Proxy server resides browser and the web server and would cache the content used by a group/set of users. Proxy servers can cache the content from web site and serve for a group of users; Proxy caching methods are discussed in [28] [29].

The caching at reverse proxies [74] enhance the throughput and response times of the web servers.

6.2.4 Web Server Level Cache
Web server can cache HTML fragments, images and static content. At web server we can also define rules to set the cache expiry headers

6.2.5 Application Level Cache
Web application uses some data objects quite frequently and they are ideal cache candidates. Data and content retrieved from database calls, service calls, lookup lists, controlled values, values from system or records (SOR) which will be frequently used across pages and user sessions would qualify for application-level caching.

Generally the application level data can be cached on client side and on the server side. One the client side, application objects can be stored in JSON objects which can be cached on the browser. On the server side, we could use caching frameworks to store the application objects on object cache. Most caching frameworks use key-value pair to store and retrieve cached objects. Using object cache we can create custom caching framework to cache the frequently used look up values, query results, search results, service responses and such. The cache TTL (Time to live) should be configured to ensure the optimal cache freshness.

6.2.6 Server Level Cache
Many application servers provide in-built caching support for caching pages. The application servers would also provide in-built cache replication across cluster nodes which can be leveraged.

6.3 Dynamic Data Caching
In this category, the cache stores the dynamic data specific for user roles, devices, pages and sessions. Dynamically cached ata is fetched at runtime using various contextual parameters such as filters, user context, user attributes, security roles, device types and such.

Caching dynamic data is has its own set of challenges. The scenario of dynamic data could arise due to multiple reasons:

• Personalized content delivery wherein the data and content on the web pages vary based on user role, access rights and authorization permissions.

• Dynamic data in database driven applications which needs to reflect updated data on web pages from database tables.

• Dynamic content coming from content management systems (CMS) and files which vary based on updates to underlying content.

Besides the above identified main categories, web page could also be dynamic due to varying page layouts (based on user role, geography and such), localization (based on geography or user preferences).

We will now discuss some of the caching techniques for handling dynamic data scenarios.

6.3.1 Database Data Caching
The most popular way to deal with dynamic data coming from database is to cache the query result, a technique known as query result caching [72]. In this technique results of popular queries are cached globally so that it could be reused across multiple users for a given user role.. Global cache can be cached for a fixed time period or can be invalidated on demand when the source data changes. Database data can also be cached in middle tier cache to reduce the load on source database.

In "trigger monitor" [77] technique, the system monitor changes to underlying data and it triggers the cache invalidation for all impacted pages using graph traversal techniques. A similar technique proposed in [84] uses dynamic content cache technique to poll the database and invalidate the stale web views/pages.

The paper [81] proposes an improvised edge computing for data intensive web sites. The "content aware caching (CAC)" technique proposed would replicate the database data locally in each of the edge servers for faster query response. Other techniques discussed in the paper are caching database records, caching query results, entire database replication and content-blind data caching (CBC) which cache the remote database query results.

6.3.2 CMS Content Caching

Dynamic content sourced from CMS is quite common in information intensive web sites. In most Common caching strategies adopted in such cases are: page level caching and fragment level caching [76] [58]. Public pages whose content would remain static for all users will be cached in its entirety. Static web pages can be cached at the web server layer or at the CDN layer. The key motivation for fragment-level caching is that a personalized page is a mix of static content and dynamic content. For instance in a typical e-commerce web site, the product content would be static whereas "purchase history" is a dynamic content based on user. In such scenarios, we will design the web page using a combination of its constituent fragments. We could cache all static content and only the dynamic content is constructed at the run time. In order to effectively cache such content, one of the popular approaches is to construct the web page through fragments [16] [17]. By breaking the page content into its constituent fragments, we can apply different cache policies at fragment level based on the dynamic nature of the content.

Edge Side Include (ESI) is a similar technique wherein a dynamic content blob is broken down into smaller cacheable content fragments [80]. These fragments can be cached at CDN layer and can be used to dynamically assemble the page at runtime.

6.3.3 Other Dynamic Caching Scenarios

Other dynamic data caching techniques include content acceleration, content-aware page caching and code caching [60] [61]. Other techniques [63] uses full-page caching for dynamic data, database table caching.

6.4 Cache Invalidation

The study [4] [25] [73] provides a list of cache invalidation algorithms. Main categories of web cache algorithm recognized in the survey [4] are traditional replacement policies, key based replacement policies and cost based replacement policies. Traditional replacement policies include Least Recently Used (LRU) which evicts least recently used object, Least Frequently Used (LFU) which evicts least frequently used object. Key based replacement policies such as LRU-MIN, LRU-Threshold, Lowest Latency first use the cache object size to evict them from the cache. Cost based replacement policies such as Lowest Relative Value, Least Normalized Cost Replacement, Size-Adjusted LRU use a cost function (usually a function of cost, size, latency) to evict the object.

Various cache invalidation policies are listed in following table 5:

Invalidation Policy	Cache Invalidation Algorithm/Method	Brief details	Challenges
Access Recency based policy	LRU, LRU-MIN, LRU- Threshold, LRU-Hot, PSS	The policy considers the last access time /last reference time period for the cache object to decide the eviction	The policy ignores the popularity, latency and other cost factors of the cache object
Access Frequency based policy	LFU, LFU-DA, LFU- Aging,	The policy considers the access frequency/popu larity of the cache object to decide its eviction	The policy ignores the recency, latency and other cost factors of the cache object
Recency and Frequency based policy	LRU*	The policy considers both recency and frequency for object eviction	
Time based policy	TTL	Each cache object is assigned a TTL value and will be automatically evicted after TTL	Suitable only for static objects with predictable change frequency. Does not considers recency, latency

On- demand invalidation	Cache invalidation service/API	Client would invoke cache invalidation API/service with cache key	Requires manual invalidation API/service invocation
Size based policy	SIZE, LRU-Min, LRU-Threshold, LRU-SP, Pitkow/Recke´rs strategy	The policy considers the size of the cache object to decide its eviction	The policy ignores the recency, latency and other cost factors of the cache object
Event based policy		A pre-specified change would automatically trigger cache invalidation	
Modification Time based policy		The policy uses the modification time for eviction	
Cost function based policy	Lowest Relative Value, Least Normalized Cost Replacement, Size-Adjusted LRU, GD(Greedy Dual)-Size, GDSF, GD*	The policy uses a cost function (usually a function of cost, size, latency) to evict the object.	Appropriate cost function has to be identified for a given context
Artificial Neural-Network based policy	Cobb & Elaarag (2008)[32] Ali & Shamsuddin (2009)[33] ElAarag & Romano(2009) [34] Sulaiman et al. (2008)[35] Tian *et al.* (2002)[36]	The eviction policy undergoes training and uses fuzzy logic to these identify the replacement candidate and to create the cache eviction policy. The algorithms would also consider the past access history to predict the likelihood the object's future access to decide its eviction	Training could be time consuming process and some of algorithms ignore the recency, latency and other cost factors
Latency based policy	Lowest Latency first	The policy considers the download latency of the cache object to	The policy ignores the recency and other cost factors of the cache object
Random policy	RAND, HARMONIC, LRU-C, LRU-S	The policy evicts the objects randomly	

Table 5. Cache Invalidation Policies

For measuring the effectiveness of cache invalidation policies, performance metrics such as hit rate, byte hit rate, delay savings ratio.

6.5 Caching Headers

At the web server layer, we can configure few parameters to improve the page performance:

• Cache headers can be set for binary/static assets based on update frequency. "Expires", "Last-Modified", "max-age" and "Cache-Control" headers can be leveraged for this.

• Cache expiry can set for the static assets based on mime types.

• We can configure performance accelerators such as mod_pagespeed for optimal performance.

7. Pre-fetching

Pre-fetching process would anticipate the need for the content keep the content ready before the actual request happens. Pre-fetching naturally would reduce the costly real-time resource calls, reduces the latency and improves the performance. Often the prefetched content is stored in cache for efficient access. A combination of efficient caching along with pre-fetching would be very effective method as discussed in [22], [23], [24]. Most effective prefetching algorithms predict the objects which would be accessed in the future and pre-fetches them.

Prefetching can be done at various layers in the request processing pipeline [4] [62] [65]: browser pre-fetch from web server, Web proxy server pre-fetch from web server, or browser pre-fetch from web proxy server. For pre-fetch between browser and web server, models such as Prediction-by-Partial-Matching (PPM) data compressor [13], resource reference patterns, user's historical access pattern [10] can be used to predict future access. Between web proxy and web server, most of the techniques [11], [12] propose pushing of content from web server to the proxy server. Another technique is to understand the common users' interests and access patterns through the proxy logs and then pre-fetch the most popular content. Similarly browsers can also pre-fetch content from proxy servers and few proxies use PPM to push the content to browser clients.

The origin servers (such as application servers) can also pre-fetch the content and assets using the access logs and content popularity.

There are other prefetching techniques such as follows:

• Association rules based prefetching uses pages are grouped based on the likelihood of intra-group resources being accessed together [14]. Association rules consider the order of page access in a user session, adjacency of pages, recency of pages to group the pages and prefetch them.

• Markov prediction model [43] uses history of users' access sequence for predicting the future access probability. The technique uses Markov model in Web application context using application access pattern

• Use Prediction by partial match (PPM) to predict the content that will likely be accessed by user using the historical access data [13]. PB-PPM is a variant of PPM which will use the subset of historical access data by using popularity (frequency of access) as the filter

• A dependency graph (DG) is constructed using the web pages as nodes and a weighted edge between nodes indicating the next access probability. The dependency graph can be used for pre-fetching using a probability threshold [30] [31]

• The survey [38] also lists other cost functions such as popularity function (which uses object popularity for prefetching) [37], object lifetime based [38] prefetch which prefetches the object with longest lifetime, good fetch which uses a combination of popularity and update rate for prefetch.

• The Survey [39] [40] identifies a clustering techniques which groups the pages based on similarity of the content, web navigation graphs, cube model, page ranking and other such clustering techniques. Page content belonging to the cluster will be prefetched. The clustering technique [24] clusters of web pages using the web user's access patterns.

• Stochastic Petri Nets (SPN) [44] based pre-fetching also considers metrics such as hit ration, byte hit ratio, latency and throughput.

• Other prefetching techniques include intelligent prefetching [45] which would pre-fetch the content to the user's nearest location, web log analysis [46] to predict web user's future requests, semantic pre-fetching [47][48] which uses document semantics for predicting future requests, location aware pre-fetching [49] which pre-fetches relevant data based on access location

7.1 Issues with Pre-fetching

Though pre-fetching improves the overall web performance, it should be noted that a sub-optimal pre-fetching strategy would result in following issues:

• Number of resource requests and network bandwidth consumed would increase due to frequent pre-fetch activity. If the pre-fetched objects are not used by the web pages, then the time, bandwidth and memory spent on those pre-fetched objects will be wasted.

• Due to frequent pre-fetch activity, the load on origin server will increase.

• In some scenarios, it is difficult to predict user's navigation behavior and web access pattern which results in unnecessary bandwidth consumed for pre-fetching objects.

In light of these challenges, we should carefully evaluate the applicability of pre-fetch schemes for a given context and test the performance metrics to evaluate the effectiveness of chosen pre-fetch strategy.

8. Infrastructure Optimization

For an optimal web performance, the underlying supporting infrastructure should be fine-tuned to ensure that it supports the expected performance.

8.1 Infrastructure Sizing and Planning

We need to properly plan for infrastructure components for optimal web performance. This involves optimal sizing and capacity planning, network planning, load balancer setup, disaster recovery (DR) setup and such [19]. Normally the infrastructure planning exercise involves expected user load, transaction rate, expected growth rate, total data and content volume and other non-functional SLAs (such as performance, scalability, availability and security).

8.2 Load Balancer

At load balancer level it is possible to use techniques such as DNS load balancing [18] (using round robin algorithm or network proximity based address resolution) and adaptive TTL [21] wherein the TTL is assigned based on client's request rate.

Another technique for load balancing is to use a dispatcher along with DNS routing [18] to route the request to one of the back end servers for optimal workload distribution transparently to the client.

Static load balancing such as round robin and central manager algorithm uses static processing information about individual nodes for workload distribution and Dynamic load balancing algorithms such as central queue algorithm, local queue algorithm constantly monitors the changing workload for a given node and uses it for workload distribution.

8.3 Server Configuration

Web server and application servers need to be properly configured for optimal performance. This include configuration of the heap memory, turning off un necessary modules, fine tuning the thread settings, configuring connection pool settings, configuring cache and expiry headers, leveraging cache plugin modules (such as mod_cache for Apache Web server). Each product vendor [20] provides a set of recommended settings and configurations for their product which can be leveraged for fine-tuning the settings.

8.4 Content Delivery Network (CDN)

CDN is a distributed network of servers which can forward-cache content and assets and serve them to the client from the nearest node in the network [18] [79]. CDNs can efficiently cache global assets such as images, multi-media files and other binary files and would optimize the perceived response time and availability of the site. It would also reduce the user load on the origin server by offloading and forward caching content and adopt smart request routing algorithms to serve the content from CDN server which causes minimal latency. CDN offers multi-geo nodes, which can optimally serve the page assets. We can use the CDN network to forward cache the images, static pages, videos, JS, CSS, JSON and such binary content. This can be used to achieve optimal performance across various geos.

CDN systems employ various surrogate servers (also known as edge servers) and request routing servers to serve the content based on its geographical proximity, reliability, network speed, data transmission cost and various such factors. The main performance metrics [79] to measure the success of CDNs are cache hit ratio, saved bandwidth, latency, server utilization and reliability. Other business success metrics for CDNs are cost saved, availability, and performance.

We also need to adopt a suitable cache invalidation mechanism to ensure that CDN cache is cleared once the asset republished.

9. Performance Monitoring

In this section we will mainly look at the performance monitoring tools from various dimensions. Performance is more of a continuous journey than an end goal. Performance has to be continuously maintained and this can happen through effective monitoring infrastructure.

9.1 Application Monitoring

During development and testing phases application monitoring mainly involves monitoring key parameters such as memory consumption rate, possible memory leaks, disk I/O, application response time. Profiling tools such as JProfiler can also be used to profile the application during development phase.

Once the application is deployed, real-time cross geo monitoring tools can be used to monitor the performance SLAs in real-time. The monitoring tools will continuously monitor the performance metrics and can be configured to automatically notify administrators if the metrics falls below the specified threshold.

9.2 Server Monitoring

Some of the open source tools such as Nagios (https://www.nagios.org/) can be used for server health check monitoring including CPU monitoring, memory monitoring, disk usage monitoring, network usage monitoring and such. Normally the server resources are closely monitored during performance testing at various load levels during performance testing. Key parameters such as response times, throughput, and resource utilization will be monitored.

Health check monitoring infrastructure also involves monitoring the uptime and availability of internal systems through heartbeat monitoring, polling and such methods.

Other monitoring tools such as real user monitoring (RUM) tools can be used for passive monitoring of user interaction with web site and end-user experience monitoring tools can be used to monitor the performance perceived by end user.

10. Performance Modeling and Testing

In this section we will look at the key concepts in performance modeling, testing and performance metrics.

10.1 Performance Modeling

Performance modeling [94] is done in the early phases of the application development. Performance modeling is an important aspect of performance based design. Main activities involved are:

• Identify performance objectives and SLAs such as response times, resource utilization and such

• Identify the main performance scenarios. This includes the popular use cases, business-critical processes, frequently accessed process steps and such

• Identify the workload (such as user load, data volume, transactions/second and such) for each scenario

• Evaluate the model to check if it meets the specified SLAs.

Performance modeling helps us to keenly analyze the performance issues in various user paths and take pro-active measures to fix it. It also helps us to validate the design from performance stand point.

10.2 Performance Testing

Performance testing is used to test the conformance of the application to the specified performance SLAs under various load conditions. During performance testing, a performance testing tool is used to load the application and various performance metrics and system resources (such as the server response time, throughput, and resource utilization) are monitored. The key performance tests are given below:

• **Load Testing:** In this scenario, we use the normal load and peak load to test the web application. The application's response time and system resource are monitored

• **Stress Testing:** The application is subjected to above peak load condition to check the behavior of the application.

• **Endurance Testing:** Normal load testing is conducted for extended duration (normally upto 72 hours) to check for resource utilization and memory leaks.

10.3 Testing Metrics and KPIs used in WPO
Given below are the main metrics used in measuring the WPO [71] [90]

• Page response time (PRT) is the overall time taken for rendering the page DOM. This includes the DNS resolution time, TCP connection time, TTFB and DOM rendition time. Normally the response times is measured at various user loads such as average load, peak load and such to measure the web site behavior.

Page response time = DNS look up time + TCP Connect time + time needed for sending, waiting and receiving + DOM load time + Render time.

At the end of DOM load time page DOM is loaded and at the end of render time the page is fully functional with all page behavior available to the user. Page will be fully interactive post render time.

• Time to first byte (TTFB) is the time taken for first byte of the response to reach the browser. TTFB is the measure of the total wait time till the browser receives the response from the server.

• **Perceived Response Time:** Also known as perceived load time, perceived load time is the page load time perceived by the end user. In most scenarios, it is equivalent to the DOM load time as user can see the key information on the page. We can optimize perceived load time by loading the key content in the beginning and injecting the JavaScript based behavior during render phase. Partial page refresh and on-demand loading, lazy loading, asynchronous loading are other techniques used to optimize perceived load time.

• **Overall Page Size:** This is the total page size which includes the size of all constituent HTML content, assets, JS, CSS, images and other elements. Naturally a bigger sized page would negatively impact the overall page performance.

• **Asset Size:** This is the sum total of size of all static assets on the page. This includes the size of all images, multi-media files, JavaScript files, CSS files, JSON files, XML files and such. As assets contribute heavily to the overall page size thereby affecting the performance, this metric is closely watched for optimization.

• **Asset Load Time:** This presents the total time taken by all assets on a web page. We can also check the individual load time and size for each of the assets to analyze the performance of assets.

• **Resource Requests:** We would measure number of resource requests required for the complete page rendition. This is the sum total of image requests, file requests, asynchronous requests and others. High number of synchronous resource requests would block the page and impact the page size and page load times.

Various page load times are depicted in Figure 5.

Web performance metrics play a crucial role during performance testing of the applications. The testing team would simulate the user load and measure

• Total user load

• Concurrent user load

• Performance SLAs

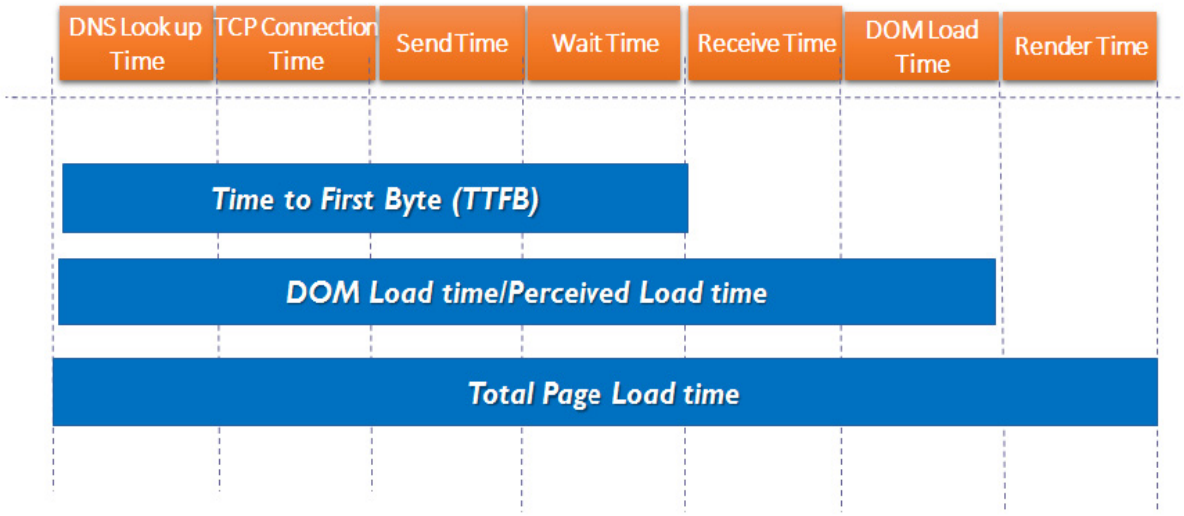

Figure 5. Page Load Times

• Transactions per second

At the system level, we would also like to test various metrics during performance testing phase. This is to ensure that the planned infrastructure can meet the performance SLAs. The key system metrics from this perspective are given below. We can use system monitoring applications to track these metrics in real-time.

• Server Throughput

• Server response time

• Server Resource utilization (CPU, Memory, disk) at various loads

• Network bandwidth utilized

Though the system metrics are mainly used for web server, it is recommended to measure this for all the systems which are involved in processing the web request pipeline (such as application server, database server). After DNS resolution and TCP connection times, the user agent would wait for the server to respond and hence it is necessary to test all the server systems involved in the processing the request.

Real-time application monitoring tools and web analytics tools can be used to measure other metrics which are indirectly impacted by page performance. We could look at following metrics to measure the impact of page performance:

• Site traffic

• Conversion rate

• Repeat visitor frequency

• Bounce rate

• Exit rate

• End user latency

11. Summary of Wpo Techniques

Table 6 summarizes the high level WPO techniques. The table provides a 2-dimensional analysis of the WPO technique on the impact on performance.

WPO Techniquet	Request Optimization	Page Size Optimization	Page Load Time Optimization	Asset Optimization	Performance at Peak load
Asset caching	Yes	No	Yes	Yes	Yes
Infrastructure Sizing	No	No	No	No	Yes
Pre-fetching	Yes	No	Yes	Yes	No
Performance Monitoring	No	No	No	No	Yes
Performance Testing	No	No	No	No	Yes
Asynch ronous	No	Yes	Yes	Yes	Yes
loading Multi-layer caching	Yes	Yes	Yes	Yes	Yes

Table 6. WPO Techniques summary

12. Web Performance Optimization Trends & Future Direction

Main trends and future direction in web performance optimization areas are given below:

• **HTML 5 Performance Enhancements:** The latest HTML 5 brings many performance enhancement features such as offline storage, hardware acceleration, JavaScript profiling, video support and other features can be used to build high performance web applications.

• **Automatic Performance Rules Execution at Runtime:** Proxies such as Google FlyWheel [50] would automatically compress the transferred data. Apache mod_pagespeed [52] module would automatically implement the web performance best practices such as content minification, asset placement, merging and such through filters.

• **Performance Friendly Protocols:** HTTP/2, SPDY protocol [51], ASAP [53], TCP pre-connect [54], TCP Fast Open [55] are used for faster speed.

• **SSL Web Acceleration:** For SSL traffic, CDNs offer SSL acceleration, SSL offloading and dynamic content acceleration modules. Leveraging P2P along with CDN [83].

• **Agile Web Frameworks:** Web frameworks such as Play framework are event-driven and provide stateless web framework. They use REST-based services for interaction and provide modular and scalable architecture.

• **Client-side MVC Frameworks:** Frameworks such as AngularJS, NodeJS, Backbone, Ember, KnockOut, provide a client-side based MVC framework for building Omni-channel, light-weight web pages. These frameworks can be used to build responsive single page applications (SPA). The performance of these web frameworks can be easily optimized due to their light-weight design.

• **Microservices Architecture:** Using Microservices architecture it is possible to build a web application as a composition of multiple independently scalable services. The architecture uses light-weight communication mechanism and mainly use functional model for building services. Since each of the microservices are individually scalable.

• **Devops Developments:** Some of the core principles of devops such as iterative performance testing, automated testing would ensure the sustained performance SLAs. Continuous Integration (CI) (such as Jenkins) is one of the key tenets of DevOps setup. CI could be used to perform iterative integration and performance testing [95]. Docker is an emerging trend in DevOps space. Docker is a self-contained container virtualization platform and it enables the microservices model. Docker helps us to individually build, test, run and deploy the services.

• **Web Oriented Architecture (WOA):** Light-weight pluggable client-side widgets are replacing server side components. WOA architecture is light-weight in design and we can easily implement the web performance best practices.

• **Application Performance Management (APM)** is a set of methodologies for end-to-end monitoring of application performance. APM can be used to detect and analyze performance bottlenecks and help in maintaining performance SLAs. APM includes various methodologies for monitoring end-user experience monitoring, analytics reporting, component-level monitoring and such.

• **No-SQL Database:** No-SQL databases such as databases storing key-value pairs, documents, graphs serve as high-performance alternatives over traditional relational databases for modern web scenarios. No-SQL database are increasingly used in Big Data applications to handle large data volume while providing high performance.

• **Cloud Computing:** More and more web applications are deployed on cloud and software as service (SaaS) is gaining huge traction for B2C application scenarios. Cloud providers would provide the infrastructure and monitoring services to track the SLAs.

• **Mobile First Development:** More and more B2C web applications are adopting mobile-first strategy. With mobile being the prime focus, performance and user experience on mobile devices will be the primary focus of testing.

• **Responsive & Adaptive Web Design:** To provide optimal performance and user experience on all mobile devices, developers are adopting responsive web design (RWD) for web development and adaptive content strategy for web content. RESS (Responsive design with Server Side components) combines RWD with server side modules to adapt to various devices. Responsive frameworks such as Twitter's Bootstrap can be used to build responsive web sites.

13. Potential Research Areas

We will look at some of the key research areas in WPO in this section.

13.1 Personalized Web Performance

Web is increasingly becoming personalized. The user experience, data, content, functionality, services are all tailor-made based on user context, user attributes and user preferences. Enhancing performance of personalized data is challenging as it is difficult to cache variation in content due to privacy and scalability concerns. There is a scope for research to create optimal balance between performance and convenience offered by personalization. Asset and data pre-fetching also would be challenging in personalized scenarios. Hence personalized pre-fetching is another candidate for research in this topic.

We saw some of the caching methods as part of dynamic data caching. However existing dynamic caching methods fall short of efficiently addressing personalization scenarios.

13.2 Mobile Web Performance Optimization

As mobile is becoming the primary target platform, optimization of web for mobile devices. Asset optimization, content rendition, user experience enhancement, progressive/adaptive enhancement, communication protocols, data exchange formats are some of the research areas in this category.

13.3 End-to-end Performance Monitoring Methods and Tools

State of the art monitoring tools mostly monitor a portion of performance processing pipeline. As a result performance engineers and system administrators need to employ multiple tools and technologies to get the complete picture of performance. The granularity of monitoring also varies across tools and technologies. Hence an end-to-end tool which can monitor all systems and components involved in the performance processing pipeline is needed.

13.4 Cloud based Performance Optimizations

Methods and techniques for optimizing cloud-based web applications is another important research area. As more and more web applications are embracing cloud phenomenon, a cloud ready web application with optimal web performance is highly desirable. This area includes research topics such as automatic performance detection in cloud, accurate performance monitoring and reporting for cloud based applications and such.

13.5 Analyzing and Debugging Performance issues End-to-end

Troubleshooting and analyzing performance issues end-to-end is challenging due to the involvement of various layers, systems and components. There is scope for development of tools to efficiently identify and analyze performance bottleneck scenarios which help in efficient troubleshooting. Tools that could pro-actively indicate potential performance bottlenecks or leading indicators/markers of a performance issues are highly desirable.

13.6 Performance Metrics and Measurement

Performance is a well-known factor which impacts user experience. It is necessary to identify various performance metrics which would impact the end-user experience and measure them efficiently. Current methods rely on couple of metrics which would not fully explain the user behavior and hence geo-specific and more accurate performance metrics is required.

It is also important to accurately measure the identified performance metrics. Existing metrics measurement techniques would not fully consider user context during measurement and reporting. As a result the performance reports do not accurately depict the performance experienced by end user. This gap needs to be addressed by simulating/considering all factors which impact the end-user performance experience in real-time.

14. Conclusion

In this paper we examined various aspects of web performance optimization. We proposed a web performance optimization framework and detailed various web performance optimization techniques such as asset optimization, caching, performance design checklist, pre-fetching, performance testing, performance monitoring.

Finally we have compiled the latest trends in WPO space and identified potential research areas in this topic.

References

[1] Google. Using site speed in web search ranking. http://googlewebmastercentral.blogspot.com/2010/04/using-site-speed-in-web-search-ranking.html, April 9, 2010.

[2] Galletta, D. F., Henry, R., McCoy, S., Polak, P. (2004). Web Site Delays: How Tolerant are Users? *Journal of the Association for Information Systems*, 5 (1).

[3] Shopzilla: faster page load time = 12% revenue increase. http://www.strangeloopnetworks.com/resources/infographics/web-performanceand-ecommerce/shopzilla-faster-pages-12-revenue-increase/

[4] Jia Wang. 1999. A survey of web caching schemes for the Internet. SIGCOMM Comput. Commun. Rev. 29, 5 (October 1999), 36-46.

[5] Chankhunthod, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., Worrel, K. J. (1996). A hierarchical Internet object cache, Usenix'96, (January).

[6] Wang, Z. (1997). Cachemesh: a distributed cache system for World WideWeb, Web Cache Workshop.

[7] Fan, L., Cao, P., Almeida, J., Broder, A. Z. Summary cache: a scalable wide-area Web cache sharing protocol, Proceedings of Sigcomm'98.

[8] Michel, S., Nguyen, K., Rosenstein, A., Zhang, L., Floyd, S., Jacobson, V. (1998). Adaptive Web caching: towards a new caching architecture, Computer Network and ISDN Systems, (November).

[9] Yang, J., Wang, W., Muntz, R., Wang, J. Access driven Web caching, UCLA Technical Report #990007

[10] Cunha, C. R., Jaccoud, C. F. B. (1997). Determining WWW user's next access and its application to pre-fetching, Proceedings of ISCC'97: *The second IEEE Symposium on Computers and Communications*, (July).

[11] Cohen, E., Krishnamurthy, B., Rexford, J. Improving end-to-end performance of the Web using server volumes and proxy _fillters, Proceedings of Sigcomm'98.

[12] Markatos, E. P., Chronaki, C. E. A TOP-10 approach to prefetching on Web, Proceedings of INET'98.

[13] Palpanas, T., Mendelzon, A. Web prefetching using partial match prediction, Proceedings of WCW'99.

[14] Cohen, E., Krishnamurthy, B., Rexford, J. Efficient algorithms for predicting requests to Web servers, Proceedings of Infocom'99

[15] Levy-Abegnoli, E., Iyengar, A., Song, J., Dias, D. Design and performance of Web server accelerator, Proceedings of Infocom'99.

[16] Challenger, J., Iyengar, A., Witting, K., Ferstat, C., Reed, P. (2000). A Publishing System for Efficiently Creating Dynamic Web Content. In *Proceedings of IEEE INFOCOM 2000*, (March).

[17] Verma, D. C. (2002). *Content Distribution Networks: An Engineering Approach*. John Wiley & Sons.

[18] Iyengar, Arun., Erich M. Nahum, Shaikh, Anees., Tewari, Renu. (2002). Enhancing Web Performance. *In*: Proceedings of the IFIP 17th World Computer Congress - TC6 Stream on Communication Systems: The State of the Art, Lyman Chapin (Ed.). Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 95-126.

[19] Killelea, P. (2002). Web Performance Tuning: speeding up the web. " O'Reilly Media, Inc.".

[20] http://httpd.apache.org/docs/current/misc/perf-tuning.html

[21] Cardellini, V., Colajanni, M., Yu, P. S. (1998). Dynamic load balancing on scalable Web-server systems. Yorktown Heights, NY: IBM T.J. Watson Research Center.

[22] Acharjee, U. Personalized and Artificial Intelligence Web Caching and Prefetching. Master thesis, University of Ottawa, Canada(2006).

[23] Huang, Y. f., Hsu, J.M. (2008). Mining web logs to improve hit ratios of prefetching and caching. *Knowledge-Based Systems*, 21 (1) 62- 69.

[24] Pallis, G., Vakali, A., Pokorny, J. (2008). "A clustering-based prefetching scheme on a Web cache environment, *Computers and Electrical Engineering*, 34 (4) 309-323

[25] Wong, A. K. Y. (2006). Web Cache Replacement Policies: A Pragmatic Approach, IEEE Network magazine, 20 (1).28–34.

[26] Chen, H. T. Pre-fetching and Re-fetching in Web caching systems: Algorithms and Simulation, Master Thesis, TRENT UNIVESITY, Peterborough, Ontario, Canada(2008).

[27] Chen, T. (2007). Obtaining the optimal cache document replacement policy for the caching system of an EC Website, *European Journal of Operational Research*.181(2) 828. Amsterdam.

[28] Kumar, C., Norris, J. B. (2008). A new approach for a proxy-level Web caching mechanism, *Decision Support Systems, Elsevier*, 46 (1) .52-60.

[29] Kumar, C. (2009). Performance evaluation for implementations of a network of proxy caches, *Decision Support Systems*, 46 (2) 492-500.

[30] Domenech, J., Gil, J. A., Sahuquillo, J., Pont, A. (2010). Using current web page structure to improve prefetching performance, *Computer Network Journal*, 54 (9) 1404-1417.

[31] Padmanabhan, V. N., Mogul, J. C. (1996). Using predictive prefetching to improve World Wide Web latency, *Computer Communication Review*, 26 (3) 22–36.

[32] Cobb, J., ElAarag, H. (2008). Web proxy cache replacement scheme based on back-propagation neural network, *Journal of System and Software*, 81 (9) 1539-1558.

[33] Ali, W., Shamsuddin, S. M. (2009). Intelligent Client-side Web Caching Scheme Based on Least recently Used Algorithm and Neuro-Fuzzy System", The sixth International Symposium on Neural Networks (ISNN 2009), Lecture Notes in Computer Science (LNCS), Springer-Verlag Berlin Heidelberg , 5552, p. 70–79.

[34] ElAarag, H., Romano, S. (2009). Improvement of the neural network proxy cache replacement strategy, *In:* Proceedings of the 2009 Spring Simulation Multiconference,(SSM'09), San Diego, California, p. 90.

[35] Sulaiman, S., Shamsuddin, S. M., Forkan, F., Abraham, A. (2008). Intelligent web caching using neuro computing and particle swarm optimization algorithm, *In:* Proceedings of the 2008 Second Asia International Conference on Modelling & Simulation (AMS 08), IEEE Computer Society, *p*. 642-647.

[36] Tian, W., Choi, B., Phoha, V. V. (2002). An Adaptive Web Cache Access Predictor Using Neural Network". Proceedings of the 15th international conference on Industrial and engineering applications of artificial intelligence and expert systems: developments in applied artificial intelligence, Lecture Notes In Computer Science(LNCS), Springer- Verlag London, UK 2358. 450-459.

[37] Markatos, E. P., Chronaki , C. E. (1998). A Top-10 approach to prefetching on the Web, *In:* Proceedings of INET'98 Geneva, Switzerland, p. 276-290.

[38] Jiang, Y., Wu, M.Y., Shu, W. (2002). Web prefetching: Costs, benefits and performance, *In:* Proceedings of the 11th *International World Wide Web Conference*, New York, ACM, (2002).

[39] Tang, N., Vemuri, R. (2005). An artificial immune system approach to document clustering, *In:* Proceedings of the Twentieth ACM Symposium on Applied Computing. SantaFe, New Mexico, USA , 918-922.

[40] Ali, W., Shamsuddin, S., Ismail, A. (2011). A survey of web caching and prefetching, *Int. J. Adv. Soft Comput. Appl.* 3 (1) 18–44

[41] Liu, M., Wang, F., Zeng, D.,Yang, L. (2001). "An Overview of world wide Web Caching", International conference on Systems Man and Cybernetics, IEEE, p. 3045-3050.

[42] Barish, G., Obraczka, K. (2000). World Wide Web Caching: Trends and Techniques.

[43] Alexander P. Pons. (2005). "Improving the performance of client web object retrieval", *Journal of Systems and Software*, 74 (3).

[44] Lei Sh., Ying-Jie Han, Xiao Guang Ding., Lin Wei. (2006). An SPN based Integrated Model for Web Prefetching and Caching, *Springer Journal of Computer Science and Technology*, 21 (4).

[45] Sajid Hussai., Robert D. McLeod. Intelligent Prefetching at a Proxy Server, *In:* Proceedings IEEE Conference on Electrical and Computer Engineering, Cananda.

[46] Yi- Hung, Arbee L.P. Chen. (2002). Prediction of Web Page Accesses by Proxy Server Log, *ACM Journal of World Wide Web*, 5 (1).

[47] Cheng-Zhong Xu, Tamer I. Ibrahim. (2003). Towards Semantics-Based Prefetching to Reduce Web Access Latency, *In:* Proceedings IEEE Computer Society Symposium, SAINT'03, U.S.A.

[48] Cheng- Zhong Xu, Tamer I. Ibrahim. Semantics-Based Personalized Prefetching to Improve Web performance, *In:* Proceedings IEEE Conference on Distributed Computing System, U.S.A.

[49] Kirchner, Holger., Krummenacher, Reto., Edwards, David., Rissel, Thomas. (2004). A Location-aware Prefetching Mechanism,

Project work at Distributed Information Systems Laboratory LSIR.

[50] Agababov, V., Buettner, M., Chudnovsky, V., Cogan, M., Greenstein, B., McDaniel, S., Piatek, M., Scott, C., Welsh, M., Yin. B., Flywheel. (2015). Google's Data Compression Proxy for the Mobile Web. *In:* Proceedings of NSDI.

[51] The Chromium Projects. SPDY. https://www.chromium.org/spdy, 2015.

[52] mod pagespeed. http://www.modpagespeed.com/.

[53] Zhou, W., Li, Q., Caesar, M.,a Godfrey. B. (2011). ASAP: A LowLatency Transport Layer. *In:* Proc. of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT).

[54] TCP pre-connect. http://www.igvita.com/2012/06/04/chrome-networking-dns-prefetch-and-tcppreconnect/.

[55] Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A., Raghavan, B. (2011). TCP Fast Open. *In:* Proc. of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT).

[56] Lohr, S. (2012). For Impatient Web Users, an Eye Blink Is Just Too Long to Wait. http://www.nytimes.com/2012/03/01/technology/ impatient-web-users-flee-slow-loading-sites.html, (March).

[57] Souders, S. (2009). Velocity and the bottom line. http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html, (July).

[58] Netravali, R., Mickens, J., Balakrishnan, H. (2016). Polaris: faster page loads using fine-grained dependency tracking. *In* 13*th USENIX Symposium on Networked Systems Design and Implementation* (*NSDI* 16).

[59] Datta, A., Dutta, K., Fishman, D., Ramamritham, K., Thomas, H., VanderMeer, D. (2001). A Comparative Study of Alternative Middle Tier Caching Solutions to Support Dynamic Web Content Acceleration. *In*: Proceedings of 27th International Conference on Very Large Data Bases (VLDB), (September).

[60] Datta, A., Dutta, K., Ramamritham, K., Thomas, H. M., VanderMeer, D. E. (2001). Dynamic Content Acceleration: A Caching Solution to Enable Scalable Dynamic Web Page Generation. *In*: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD), (May).

[61] Iyengar, A., Challenger, J. (1997). Improving Web Server Performance by Caching Dynamic Data. *In*: Proceedings of Usenix Symposium on Internet Technologies and Systems, (December).

[62] Zukerman, I., Albercht, D., Nicholson, A. (1999). Predicting Users' Requests on WWW. *In:* Proceedings of 7th International Conference on User Modeling, (June).

[63] Zhu, H., Yang, T. (2000). Cachuma: Class-based Cache Management for Dynamic Web Content. Technical Report TRCS00-13, Dept. of Computer Science, The University of California at Santa Barbara, (June).

[64] Su, Z., Yang, Q., Lu, Y., Zhang, H. (2000). WhatNext: A Prediction System for Web Requests using N-gram Sequence Models. *In*: Proceedings of 1st International Conference on Web Information System and Engineering, (June).

[65] Wang, Z., Crowcroft, J. (1996). Prefetching in World Wide Web. *In:* Proceedings of IEEE Global Telecommunications Internet Mini-Conference, (November).

[66] Souders, S. (2007). High performance web sites: Essential knowledge for frontend engineers. Farnham: O'Reilly.

[67] Souders, S. (2009). Even faster web sites. Sebastopol: O'Reilly.

[68] Sundaresan, S., Magharei, N., Feamster, N., Teixeira, R. (2013). Characterizing and Mitigating Web Performance Bottlenecks in Broadband Access Networks.

[69] Cohen, E., Kaplan, H. (2001). Proactive caching of DNS records: Addressing a performance bottleneck. *In:* Symposium on Applications and the Internet (SAINT) p. 85–94.

[70] Feldmann, A., Caceres, R., Douglis, F., Glass, G., Rabinovich, M. (1999). Performance of web proxy caching in neous bandwidth environments. *In:* Proc. IEEE INFOCOM, New York, NY, (March).

[71] Palmer, J. W. (2002). Web site usability, design, and performance metrics. *Information Systems research*, 13 (2) 151-167.

[72] Yagoub K, Florescu D, Issarny V, Valduriez, P. (2000). Caching strategies for dataintensive Web sites. *In:* Proceedings of the 26th international conference on very large data bases, (May).

[73] Podlipnig, S., Böszörmenyi, L. (2003). A survey of web cache replacement strategies. *ACM Computing Surveys* (CSUR), 35 (4) 374-398.

[74] Iyengar, Arun., Rosu, Daniela. (2002). Architecting Web sites for high performance. *Sci. Program.*10, 1 (January), 75-89.

[75] Adali, S., Candan, K. S., Papakonstantinou, Y., Subrahmanian, V. S. (1996). Query caching and optimization in distributed mediator systems. In ACM SIGMOD Record 25 (2) 137-146. ACM. Chicago, (June).

[76] Challenger, J., Iyengar, A., Witting, K., Ferstat, C., Reed, P. (2000). A Publishing System for Efficiently Creating Dynamic web Content Proceedings of IEEE INFOCOM.

[77] Challenger, J., Iyengar, A., Dantzig, P. A Scalable System for Consistently Caching Dynamic Web Data, *In:* Proceedings of IEEE INFOCOM'99

[78] Apostolopoulos, G., Peris, V., Saha, D. (1999). Transport Layer Security: How much does it really cost? *In:* Proceedings of IEEE INFOCOM.

[79] Vakali, A., Pallis, G. (2003). Content delivery networks: Status and trends. *Internet Computing, IEEE*, 7 (6) 68-74.

[80] Ravi, J., Yu, Z., Shi, W. (2009). A survey on dynamic Web content generation and delivery techniques. *Journal of Network and Computer Applications*, 32 (5) 943-960.

[81] Sivasubramanian, S., Pierre, G., VanSteen, M., Alonso, G. (2007). Analysis of caching and replication strategies for Web

applications. *IEEE Internet Comput* 11 (1) 60–6.

[82] Yagoub, K., Florescu, D., Issarny, V., Valduriez, P. (2000). Caching strategies for dataintensive Web sites. *In*: Proceedings of the 26th international conference on very large data bases, (May).

[83] Fortino, G., Mastroianni, C. (2008). Special section: enhancing content networks with P2P, grid and agent technologies. *Future Gener. Comp. Syst.* 24 (3) 177–9.

[84] Candan, K. S., Li, W. S., Luo, Q., Hsiung, W. P., Agrawal, D. (2001). Enabling dynamic content caching for database-driven web sites. *In:* ACM SIGMOD Record. 30 (2) 532-543. ACM. (May).

[85] Best Practices for Speeding Up Your Web Site: http://developer.yahoo.com/performance/rules.html

[86] Ravi, Jayashree., Yu, Zhifeng., Shi, Weisong. (2009). A survey on dynamic Web content generation and delivery techniques. *J. Netw. Comput. Appl.* 32 (5) (September), 943-960.

[87] Web performance optimization: http://en.wikipedia.org/wiki/Web_performance_optimization

[88] For Impatient Web Users, an Eye Blink Is Just Too Long to Wait: http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html?_r=2

[89] Akamai Report: http://www.akamai.com/html/about/press/releases/2009/press_ 091409.html

[90] Rempel, G. (2015). Defining Standards for Web Page Performance in Business Applications, *In:* Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering - ICPE '15.

[91] Galletta, D. F., Henry, R., McCoy, S.,Polak, P. (2004). Web Site Delays: How Tolerant are Users?, *Journal of the Association for Information Systems*: 5 (1) Article 1.

[92] Hoxmeier, J. A., DiCesare, C. (2000). System response time and user satisfaction: an experimental study of browser based applications. Proceedings of the Americas Conference on Information Systems, (Long Beach, CA, USA, 2000), Association for Information Systems, 140-145.

[93] Google Page Speed Insights and Rules: https://developers.google.com/speed/docs/insights/rules

[94] Kambhampaty, S., Modali, V. S., Bertoli, M., Casale, G., Serazzi, G. (2005). Performance Modeling for Web based J2EE and .NET Applications, *In:* Proc. of world Academy of Science, Engineering and Technology, 8, (October).

[95] Barker, T. (2014). High performance responsive design: Building faster sites across devices (1st ed.). O'Reilly Media.