International Journal of Computer Trends and Technology
ISSN: 2231 – 2803 / https://doi.org/10.14445/22312803/IJCTT-V69I8P105

Volume 69 Issue 8, 20-22, August 2021

Original Article

Designing for Latencies

Anoop Koloth

Founder, Dotted yellow, San Jose, United States of America.

Received Date: 10 July 2021
Revised Date: 13 August 2021
Accepted Date: 26 August 2021

Abstract *- When an organization spends CAC (Cost of Acquisition) to acquire a potential customer through various channels, all it takes is the LTV (Life Time Value) of the customer to decide the future of the upcoming company and its growth initiatives.*
Ideally, we would expect the LTV > CAC for the business to thrive and grow or acquire a more extensive user base for future anticipated growth.

However, when the customers have unlimited choices at their fingertips to choose from, it becomes imperative to design the product for a better user experience and latency compared to the competitor.

The paper looks at various design approaches to latency and how the technology landscape is changing around latencies.

Keywords *- Latencies, Caching, Optimization, Web, Mobile, Edge Computing, User perception.*

I. INTRODUCTION

Latency, in simpler words, is the time taken for data transfer from source to destination measured typically in milliseconds. And here, we are referring latencies across three central user platforms: websites, mobile websites, and Native Mobile applications. To begin with, let's understand what introduces latencies into the system. The current technology stack in the industry is multi-tiered with the predominant pattern as below

Client – CDN – Gateways – Compute – Data Stores

And any functionality to be addressed for the end user would mean a hop in the bidirectional to get client requests, process the function, associate the data, and give back response to stitch the end-user experiences. That meant that requests and responses exhibit packets' lifecycle traversing across the internet or network with multiple hops, thereby introducing delays or latencies.

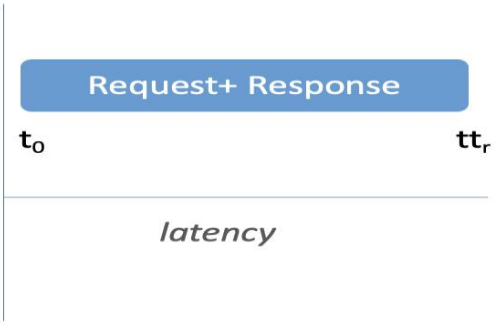
During the advent of the Web, the initial static Html or assets, a single network request would suffice the end-user pageview or experience. Which would primarily be impacted by how much time the client is spending on the domain name resolution, secure service layer computation, actual packets transfer of data from source to destination. The primary governing criteria would be the route taken over the internet and the proximity of the data center, all attributed to the request and response time and network latencies.

However, with the dynamic content powered by JavaScript on the Web and inquisitive styling by the cascaded style sheet, latencies are no longer governed by just one request-response, or several hops or number of requests made. It has become a conglomerate of various factors.

Primarily the below four factors are the ones that have a higher impact on latency for any software system on the internet. And pretty much any component exhibits a fractal pattern on the below four parameters into consideration during the bottleneck phase.

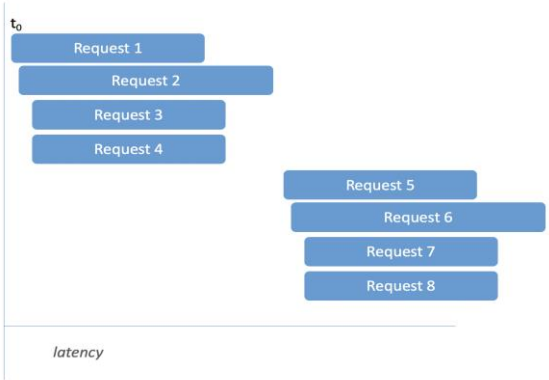
- ***Time to Response(tt_r)***

The time taken to accommodate a network request or a response to compute.



- ***Multiplex Request Calls(mi)***

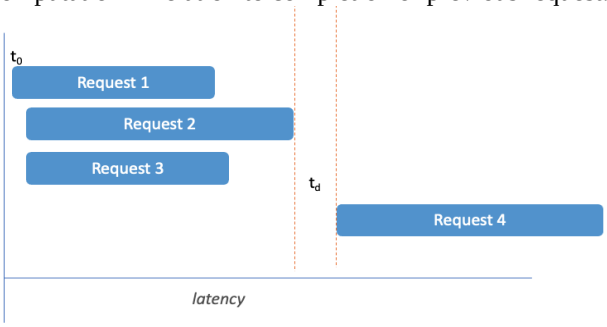
The different requests required to accommodate an experience and their parallelism with each other.



In the above diagram the value for mi is 4. Typically, its averaged across batch of parallel network calls or system requests over the time window.

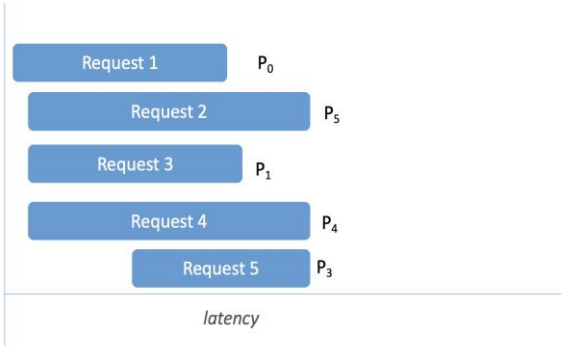
- ***Delay to Start(t_d)***

The delay to start a subsequent critical request or computation in relation to completion of previous request.



- ***Prioritization (P_o)***

The order of execution of network request or the computation.



$$P_o = \sum(P_i * tt_{rx})/N$$

N = Total number of requests

tt_{rx} = Time to response for specific request

- ***Bottleneck(B)***

The factors impending delays at different layers in OSI [1] model. Typically, bottleneck is categorized as

$$B_x = f(\min(P_0), \min(t_d), \min(tt_r), \max(mi))$$

The goal is to reduce the Bx at each iteration and tier to arrive at the optimal latency experience.

II. DESIGN

Designing an efficient latency system aims at minimizing Bx at each component or tier, or layer. However, to survive and thrive against competitive user experiences, the industry has resorted to primarily two techniques “Trick” or “Treat” or a “Hybrid” of both. In the “Trick” scenario, the aim is to the illusion that the page or experience is fast, or delay goes unnoticed. The Treat side of affairs, it focuses mainly on minimizing the Bx. Another design pattern into consideration is optimizing latency w.r.t to Space, Time, and State.

Let us take a deeper look into the individual experiences and see how the design is leveraged.

III. WEBSITES

Most of the internet was Web-based for more than a decade before the advent of mobile devices and the internet of things. Basic HTML-driven pages powered the experiences, then came the dynamic ones powered by JavaScript and styled by cascading style sheets. As the aspiration for better-looking web pages and experiences to attract customers and beat competitors kept growing, so did the static assets and associated new style sheets and JavaScript libraries powering the gigantic Content Driven networks. The game changed for CDN when images and videos became a predominant asset category with the advent of social media and networking.

It all started with server-driven architecture with Client thin design, eventually phasing out through server and thick client design. We are now endeavouring to a multi-tiered or multi geographies and region-driven architecture with data split across different tiers to power the consolidated experiences to the end-user.
Also, the industry has resorted to another conglomerate approach to the problem stitching experiences across different touchpoints or platforms. Over time, what hasn't changed is the end-user perception to get things quickly and most intuitively.

A. Trick

Let us go over a few of the industry-wide tricks to beat delays in user perception.

a) Changing the Wait experiences

Here, the basic methodology is to hide the backend delays to the end-user by either displaying a silhouette with blended background colours or giving an informative animation that engages the end-user to delay the realization phase eventually.

This method buys time from the user in the background to address all the underlying issues like server-side latencies, DNS times, assets load times, critical resource failures, bandwidth constraints, etc. This technique should refrain as this still doesn’t solve the root cause of higher Bx. But this is a win-win solution when the latencies degradation is due to network issues or external third-party failures beyond the scope of optimization.

b) Above the Fold and Below the Fold

In any website, the experience available to the end-user before the scroll is above the fold. For faster experience, the methodology prioritizes the above-the-fold data and assets over the below-the-fold resources and data modules.

The delays incurred in loading resources are shifted to below the fold experience and buried underneath the time taken by the user to act on-page.

c) Predictive Loading & Prefetching

With many resources heavy on media files and images taking over a more significant portion of newer websites traffic, the traditional approach relying on the round trip time optimization in a transaction resulted in delayed and buffered experiences. Prominent industry players have either resorted to Edge or Predictive or pre-fetching techniques to have the data available in the proximity of the end-user devices to provide a seamless latency experience. Powered by machine learning and user data analytics, the effectiveness of the data made available has improved recently. However, this approach fails to address the long tail or new user scenarios.

d) State Management on the Browser

Browsers today power the HTTP2 and Server push technologies. The advent of stacks like React, which can trigger events on the DOM tree based on state changes, allows async data availability at the browser. Also, this gives better control to the Web Developer to manage state, thereby leveraging heavy ended resource utilization asynchronously and give seamless end-user experience without incurring delays.

B. Treat

The rule of thumb has not changed a lot at the client layer, which is modern-day Browsers for Websites. All it matters is the optimized Critical Rendering Path [2] and faster access to data.

Current-day browser experiences powered by multiple javascript, images, style sheets, and an optimized critical rendering path would infer no bottlenecks or delays in completing the Document Object Model.
Once the primary data is available for the HTML parsing, any invocation to the javascript would block the parser until the availability of javascript. Similarly, the style sheets act as the primary render blocker. Optimization aims to speed up the HTML parsing for base templating and allow the available data to render by style. Once the ttr is minimized for the primary call, which could include a series of methodologies like caching, DNS prefetch,
tls pre-termination, prefetching, or server push. The vital design consideration is minifying, modularizing, reprioritization, and reducing IO inlining, easing the browser rendering path.

IV. MOBILE WEBSITES

Mobile websites started minimalistic on the mobile devices but have been gaining traffic attributed to organizations moving to a single application model with the complexities of maintaining two native stacks. Also, the technology stack has advanced to support many native capabilities on the web. Mobile Web for the major industry players has been primarily vital in the deep linking campaign to new users to mobile app downloads. Regarding rendering optimization, the same as websites are applicable; however, due to the responsive design pattern and different devices specs, the goals are primarily to focus on the battery usage, CPU cycles, memory consumption and have minimalistic to the assets leveraged.

V. NATIVE MOBILE APPLICATIONS

Leading the internet traffic user experiences are the native mobile application with the heterogeneous spectrum of experiences ranging from gaming, virtual reality, augmented reality, social networking. Today, mobile applications are the most data-sensitive application on the planet, just behind the internet of things. Unlike the Websites, the critical rendering path is insignificant, as the assets required for rendering are part of the binary. The display experience on the native mobile OS stack is optimized.

However, the case may not be the same for mobile applications that cannot bundle the assets in the constrained binary size limit enforced by the platform provider. Mobile applications primarily depend on network performance [3] and client-side serialization and deserialization techniques implemented. The design goal is employed mainly to minimize.

VI. EDGE COMPUTING

Edge computing is primarily computing offloading at edge server closers to user proximity compared to everything on the cloud or the data center. The compute offloading, or local computing, has been around on the client devices; however, the capabilities were limited.

When client devices or browsers have moved on to accommodate more these days on local computing with the advancement of technologies like server push and state management on the client.

Edge computing powered by the recent trend in software-defined networks and software load balancers is beneficial for controlling the data. The major challenge for latency at the edge is the balance between data availability and latency improvement. The design consideration is the proximity of edge nodes, time to live for the data, tls termination, handling pre-flight browser calls, optimizing the cache purge strategies. Given the lower Capex of edge infrastructure and network optimizations like anycast systems, the data localization on edge nodes based on session affinity also does the trick to improvise data availability and latencies.

VII. CONCLUSION

Performance and user perception have become an essential aspect impacted by latencies typically considered after the product development or production deployment. Early in the product lifecycle, the latency design approach is vital for any uplift and eliminates terrible user experiences, as anyone who had one will not always report.

REFERENCES

[1] ISO: Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model. Geneva, Standard ISO/IEC 7498-1(E) (1994). [Online]. Available: https://www.iso.org/standard/20269.html
[2] W3C Recommendation: Navigation Timing. [Online]. Available: https://www.w3.org/TR/navigation-timing/
[3] A. Koloth, “Native App Network Performance” Dzone., Apr. 2021.