Beaconnect: Continuous Web Performance A/B Testing at Scale

Wolfram Wingerath*
wolle@uni-oldenburg.de
University of Oldenburg, Germany

Benjamin Wollmer*
benjamin.wollmer@uni-hamburg.de
University of Hamburg, Germany

Markus Bestehorn
bestem@amazon.ch
Amazon Web Services, Switzerland

Stephan Succo
ss@baqend.com
Baqend, Germany

Sophie Ferrlein
sf@baqend.com
Baqend, Germany

Florian Bücklers
fb@baqend.com
Baqend, Germany

Jörn Domnik
jd@baqend.com
Baqend, Germany

Fabian Panse
fabian.panse@uni-hamburg.de
University of Hamburg, Germany

Erik Witt
ew@baqend.com
Baqend, Germany

Anil Sener†
senera@amazon.com
Amazon Web Services, UK

Felix Gessert
fg@baqend.com
Baqend, Germany

Norbert Ritter
norbert.ritter@uni-hamburg.de
University of Hamburg, Germany

ABSTRACT

Content delivery networks (CDNs) are critical for minimizing access latency in the Web as they efficiently distribute online resources across the globe. But since CDNs can only be enabled on the scope of entire websites (and not for individual users or user groups), the effects of page speed acceleration are often quantified with potentially skewed before-after comparisons rather than statistically sound A/B tests. We introduce the system Beaconnect for collecting and analyzing Web performance data without being subject to these limitations. Our contributions are threefold. First, Beaconnect is natively compatible with A/B testing Web performance as it is built for a custom browser-based acceleration approach and thus does not rely on traditional CDN technology. Second, we present our continuous aggregation pipeline that achieves sub-minute end-to-end latency. Third, we describe and evaluate a scheme for continuous real-time reporting that is especially efficient for large customers and processes data from over 100 million monthly users at Baqend.

PVLDB Reference Format:
Wolfram Wingerath, Benjamin Wollmer, Markus Bestehorn, Stephan Succo, Sophie Ferrlein, Florian Bücklers, Jörn Domnik, Fabian Panse, Erik Witt, Anil Sener, Felix Gessert, Norbert Ritter. Beaconnect: Continuous Web Performance A/B Testing at Scale. PVLDB, 15(12): 3425 - 3431, 2022.
doi:10.14778/3554821.3554833

1 INTRODUCTION

While the importance of content delivery networks (CDNs) for fast page loads is generally undisputed, accurately measuring their impact is inherently difficult: Since a CDN is integrated as a reverse proxy via Domain Name System (DNS) rules, it can only be enabled or disabled for all users of a given website at once and uplift measurements are therefore only possible via before-after comparison. Even though statistically clean A/B testing is the default in many other areas of Web optimization (e.g. user interface design [46] or marketing campaign planning [5]), the best practice for evaluating page speed is naturally distorted by effects like fluctuating online activity or different marketing campaigns being active over time. We argue that skew in uplift measurements can be avoided in the context of page speed acceleration and therefore present Baqend's approach towards achieving and measuring website acceleration.

Baqend's architecture for content delivery relies on two main components: While Beaconnect is the pipeline for aggregating and reporting real-user performance data, it has been developed for use with Speed Kit [41] as the technology for page speed acceleration. Speed Kit is a novel technology for website acceleration that uses a CDN internally, but is controlled through a JavaScript library. In consequence, its configuration is more flexible than simple DNS rules pointing to a CDN: Being a browser-based approach, Speed Kit can be enabled and disabled for individual users and is therefore naturally compatible with statistically sound A/B testing. To quantify website acceleration with Speed Kit, it is enabled for only a certain share of website visitors (e.g. 50%) while data is collected from all of them. By collecting data in this fashion, it is possible to compare measurements from users on the stock website with measurements from users on the accelerated website – collected over the same timeframe and under otherwise identical conditions.

A particularly challenging aspect of operating Speed Kit is to report all relevant metrics accurately while minimizing latency and processing overhead. Tracking tools like Google Analytics [30] or Adobe Analytics [6] are unsuitable for this purpose as they rely on data sampling [28] and incur a processing latency of hours or even days [29]. Real-user monitoring solutions such as mPulse [8] or Dynatrace RUM [21] do provide low latency and avoid data sampling, but would require tight integration into Speed Kit for sufficient data coverage which is prohibitively complex.

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554833

*Wolfram Wingerath and Benjamin Wollmer contributed equally to this work.
†Author was affiliated with Amazon Web Services, Inc. during contribution period.

1.1 Contributions & Outline

We developed Beaconnect to capture Web performance with Speed Kit in realtime and A/B testing it at scale. Our pipeline handles more than 650 million individual page impressions (PIs) from over 100 million unique users monthly. We make the following contributions:

- We discuss related work in Section 2 before explaining Beaconnect's unique aptness for A/B testing and deriving requirements for simultaneously providing offline data warehouse analytics and real-time reporting in Section 3.
- In Section 4, we present Beaconnect's aggregation strategy for continuous real-time reporting with sub-minute latency.
- We finally describe our production system and present experimental results in Section 5, share developer experiences in Section 6, and conclude in Section 7.

2 RELATED WORK

In this section, we provide an overview over related work.

Page Speed & User Satisfaction. E-commerce researchers have shown that users of websites expect response times of *2 seconds or less* [17, 32]. Failure to meet this standard results in decreased user satisfaction [13], and potential business loss [44]. To meet this 2-second requirement, websites rely on content delivery networks (CDN) [35, 36] such as Akamai [1], Fastly [24], Cloudflare [2], or Amazon Cloudfront [10]: CDNs operate hundreds of endpoints distributed across the globe and each of these endpoints replicates website content that originates on the Web servers of the website. This allows serving content to users with less latency as less network traffic has to travel from the aforementioned Web servers to the user and instead is delivered by a CDN endpoint close[1] to the user. While the use of CDNs is well established, measuring their achieved performance uplift in skew-free fashion is inherently hard.

Web Performance Monitoring. Web performance monitoring refers to the process of observing the state of Web services or Web applications over time by employing either external or internal monitoring or measuring artifacts [20]. The two prevalent ways to collect performance data are to either generate *synthetic data* or to perform *real-user monitoring (RUM)*. Synthetic approaches [3, 4, 18] use artificially generated Web traffic to simulate website interactions. While synthetically generated data is easier to produce at scale, it has the potential of missing important properties of real-world behavior as it is limited to measuring purely technical metrics [20]. Business metrics such as conversion rate[2] or session duration therefore cannot be captured by synthetic approaches.

RUM uses data generated by real-user behavior to monitor a website. There is a large body of work dealing with analytics as well as data management in RUM: [19] tries to determine failure probabilities in large systems using end-to-end traces and a hierarchical clustering method. Similarly, [7] uses statistical models to find latency bottlenecks in communication paths and [15, 16] use data mining to predict performance. In many cases, the analysis occurs offline or with significant delay in the order of hours or even days which is an issue particularly in the context of split testing as we show next. Beaconnect implements RUM using data beacons emitted asynchronously by Speed Kit [41] via a light-weight JavaScript that is executed in the browser of users. Contrary to the aforementioned approaches, this beacon-based approach also allows capturing business metrics. Section 3 describes our data collection and Section 4 provides details for near-real-time analytics.

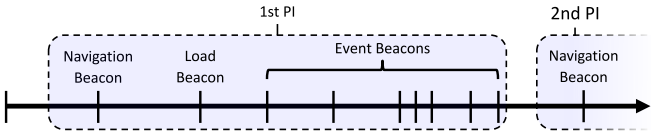

Figure 1: Beaconnect continuously aggregates data beacons into page impressions (PIs).

There are also tools like Google Analytics (GA) [30] and mPulse [8] that can be used to monitor real-user performance. Similarly to the statistical approaches mentioned above, GA samples data and thus introduces inaccuracy. Furthermore, the reporting delay for metrics can be significant depending on the traffic volume that occurs on a website [29]. Akamai's mPulse delivers performance metrics with low latency through a tight integration with the Akamai CDN [1]. However, A/B-testing CDN performance remains inherently challenging due to the inherent limitations of CDN technology mentioned in Section 1 and described below.

Web Performance & Split Testing. For feature development and deployment, A/B testing (also known as split testing) has been a de-facto standard for more than a decade in e-commerce sites [23, 33, 34, 38]: Before rolling out a new website feature for all users, the traffic is split into two groups where one group of the users gets to use the new feature and the other group keeps using the status quo. This mechanism allows measuring the effects of incremental improvements as user behavior with and without it can be compared side-by-side. For example, it is possible to determine whether a new functionality on an e-commerce website results in more sales or whether changing the position of a particular button increases the likelihood of it being clicked. The underlying requirement for split testing is that multiple versions of a website can be delivered to separate sets of randomly chosen users.

Implementing these random sets of users is challenging in the context of CDNs as they can only be enabled or disabled for all users. Theoretically, round-robin DNS load-balancing could be used to create a split test setup in which the domain of a website is associated with multiple IP addresses, some of which belong to a CDN and some of which do not. However, such a setup would be extremely difficult to control due to a number of potential side effects that are out of the CDN provider's hands (e.g. skew introduced by DNS resolvers or caches, mid-session group changes because of expiring DNS caches, or a wide range of other side effects occurring when a browser establishes more than one connection to the website). We are not aware of a single implementation of such a test setup and consider the chances of generating meaningful results slim at best.

In combination with CDNs, performance-related metrics are hence only available using a before-after approach, i.e. comparing performance metrics collected before a change was deployed with metrics from after the change. This introduces delays and makes it difficult to correlate measurable changes in performance metrics

[1] CDNs have different algorithms to find the most suitable endpoint. In general, the endpoint is chosen so that the latency is minimal.
[2] This is typically the ratio of transactions in a store vs. the number of visitors/sessions.

with a specific deployment (especially for large e-commerce websites where multiple changes occur on any given day). Beaconnect avoids this issue as it natively supports A/B testing and reports non-sampled performance and business metrics in realtime.

3 APPLICATION DOMAIN & REQUIREMENTS

In this section, we provide an overview of data collection in the browser (Speed Kit) and the required backend processing (Beaconnect) to derive relevant technical and business insights. We close the section with a brief discussion of requirements for Beaconnect's data aggregation pipeline.

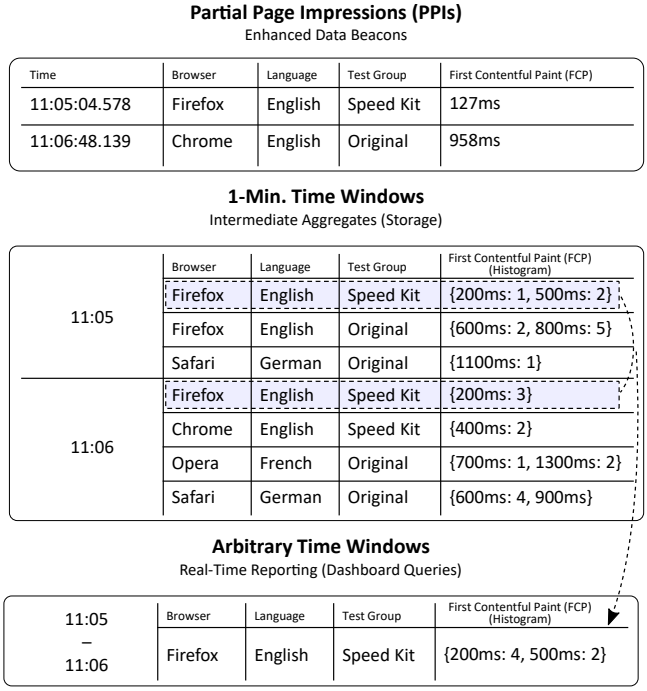

Figure 2: Beaconnect's continuous aggregation pipeline writes data summaries to an intermediate storage. Queries over this intermediate storage can be evaluated efficiently as they touch fewer records than queries over raw data would.

Split Testing Web Performance With Speed Kit. Contrasting traditional CDNs, Speed Kit [41] is a browser-based technology that makes split testing straightforward. It is designed as a code plugin for (e-commerce) websites and can be included as a JavaScript snippet in the website's HTML. Relying on the Service Worker Web standard [9], Speed Kit is implemented as a proxy within the browser that redirects certain browser requests to a specialized caching infrastructure rather than the (comparatively slower) origin Web server. On the very first website visit, the browser downloads and executes Speed Kit like any other third-party library. Speed Kit's behavior is then controlled through a configuration that is retrieved on the very first visit and updated in the background thereafter. During a split test, Speed Kit chooses one of the test groups with a probability respecting the configured split and correspondingly either activates the acceleration feature or not. By collecting data from both test groups, the effects of page speed acceleration can thus be measured in a statistically clean way.

Data Granularity: Beacons, PIs & Sessions. While page impressions (PIs) seem like the atomic unit to reason about in the context of Web performance monitoring, data is actually collected at a finer granularity within the browser. Sending multiple data beacons for every PI minimizes data loss as information is sent as early as possible, but also requires connecting all data beacons to full PIs later on and thereby increases complexity for backend processing. As illustrated in Figure 1, Speed Kit emits three kinds of data beacons on multiple occasions throughout the page load. Static information such as the target URL or the current timestamp can be sent away as soon the navigation starts (navigation beacon), whereas values obtained from the browser's Performance API [45] cannot be sent until very late in the load process (load beacon), including the time it took to load the page and the time until the first or largest contentful paint (FCP and LCP, respectively). Certain events may even occur long after the page load has completed and are therefore handled via dedicated and optional transmissions (event beacons). For example, a user may read the product description first and then put an item into the shopping cart, thus triggering a new transmission. As another example, JavaScript errors may occur at any point in time and also need to be reported.

3.0.1 Backend Aggregation: Connecting Beacons. Sending partial information on PIs obviously avoids data loss through deliberate delays, but also requires connecting all data beacons to full PIs later on and thereby increases complexity for backend processing. This principle of connecting beacons is eponymous to Beaconnect. Data beacons for the same PI usually occur in relatively short succession and no new data beacons will be generated once the user has moved onto the next page. We therefore close aggregation windows for any given PI or session after 30 minutes of inactivity which achieves good results in practice and is in line with industry best practices (e.g. Google Analytics [27]).

Requirements for Analytics & Reporting. There are two use cases for data aggregation in the context of Web performance monitoring and tuning that have largely incompatible requirements. First, the fully aggregated PIs need to be retained for exploratory analysis and debugging individual user sessions. This use case requires high data resolution, but executing queries offline is acceptable. After all, deep-dive analyses typically revolve around reported issues and rarely target data from ongoing sessions. Second, though, data aggregates (i.e. summaries) need to be reported continuously for effective operations monitoring and validating certain metrics on new deployments during and immediately after rollout: Contrasting the analytics use case, real-time reporting thus critically depends on low latency, but does not require support for arbitrary analyses. This paper is focused on the second use case, i.e. continuous aggregation that enables efficient real-time reporting. We refer to [25, 40] for details on our data warehousing approach.

4 REAL-TIME REPORTING EFFICIENCY

Intuitively, all high-level performance metrics can be computed on top of fully aggregated PIs. However, this approach incurs significant delays in reporting due to the 30-minute timeout for the PI aggregation window and an additional delay for aggregating

session-based data. To minimize the time it takes for data to become visible after collection, Beaconnect's continuous real-time summary aggregation produces output without waiting for the PI timeout.

Partial PIs: Data Beacons + PI & Session Dimensions. The basic idea is to pre-aggregate the incoming data by different attributes to speed up standard analyses via executing queries over the intermediate aggregates instead of the raw data. The continuous aggregation therefore buffers data beacons for every PI only for a short time until the attributes required for pre-aggregation have been observed: Once these *dimension attributes* have been attached to a beacon, we call this beacon a *partial PI* to make the distinction from a raw data beacon explicit. Dimension attributes that are typically stable are buffered for the entire session (e.g. browser or test group), while potentially volatile dimension attributes are buffered on PI-level (e.g. the currently selected language). Once all dimensions for a PI have been observed, all subsequent beacons of that PI contribute to intermediate aggregates without any delay.

Histogram Summaries for Performance Timers. As shown in Figure 2, all partial PIs are collected over a small tumbling window (e.g. 1 minute) and then aggregated by different dimension attributes such as browser, language, test group, and others. Although not illustrated, the intermediate aggregates also contain additional attributes such as the number of observed beacons and the number of PIs. To avoid storing a great number of raw performance timers, metrics such as the first contentful paint (FCP) are bucketed into histograms. We chose histograms for compressing raw timer values, because averages and quantiles can be computed on their basis[3]. For queries over arbitrary time windows, using the intermediate aggregates instead of the raw PI data can accelerate runtime significantly (cf. Section 5). While the intermediate aggregation does incur a certain delay, end-to-end latency is usually smaller than aggregation window size: For a 1-minute window under constant beacon inflow, for example, a beacon is only delayed for 30 seconds on average as the actual buffering time depends on when the beacon arrives and when the aggregation window closes.

5 PRODUCTION DEPLOYMENT & TRAFFIC

As of January 2022, Beaconnect handles over 3 billion data beacons per month, which corresponds to more than 650 million PIs from over 200 million user sessions and over 100 million unique users. To avoid load skew, we partition the data by user session IDs that are randomly generated in the browser. But since most of our customers are based in northern Europe, the amount of incoming data still fluctuates heavily throughout the day with a factor of about 14x between minimum and peak traffic. Scaling up and down our pipeline without service interruption is therefore required for efficiency, but also challenging as our aggregation scheme requires holding every active user session in memory.

Deployment as a Service. Our production system is hosted as a collection of fully managed services on Amazon Web Services (AWS) to enable elastic scaling [43]. We chose Flink [26] for executing our workload as it is suitable for running with large state and is a native stream processor, thus delivering better latency

[3]The bucketing in our production system is more fine-grained than illustrated in Figure 2 as it varies with size of timer values to increase precision for smaller measurements (1ms-buckets up to 500ms, 10ms-buckets up to 5s, 100ms-buckets up to 60s, and so on).

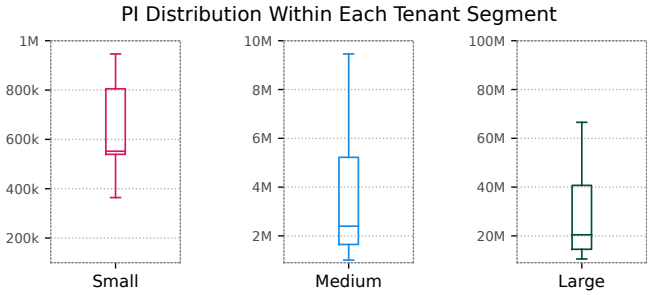

Figure 3: A Tukey box plot for the distribution of tenant sizes measured by the monthly traffic in page impressions. Median traffic in the Small, Medium, and Large tenant segments is separated by one order of magnitude each.

than systems such as Spark [47] or Spark Streaming [48] which are based on processing (micro) batches [42, Ch. 5]. The data beacons are streamed into an Amazon Kinesis [11] data stream which is then consumed by our Flink application for further processing. The first processing step is data cleaning to prepare the beacon data for downstream aggregation and write off invalid beacons for later problem analysis. The user agents [14, Sec. 10.15] are then resolved to human-readable data artifacts which are added as additional attributes such as the device type or browser name. We also monitor traffic for suspicious behavior to identify bots and scrapers: The associated user agents are stored in DynamoDB [39], so that the state required for identifying such synthetic traffic is persistent and independent of Flink snapshots. Finally, the cleaned and enriched data beacons are fed into our dual aggregation pipeline as outlined in Section 4: The intermediate summary aggregates are ingested into an Elasticsearch [22] cluster for real-time reporting via Kibana [31] dashboards, whereas the fully assembled PI data stream is persisted in S3 [12] block storage for historical data analysis and operational troubleshooting with our data warehouse built on Presto [37].

Tenants by Traffic: 3 Orders of Magnitude. The amount of beacon data being collected and processed for real-time reporting naturally depends on the actual number of user traffic on the specific tenant's website. Figure 2 illustrates how our tenants can be classified into three segments by their respective monthly PIs (just referred to as PIs throughout the rest of this section): *Large* with at least 10M PIs, *Medium* with 1M to 10M PIs, and *Small* with less than 1M PIs[4]. The *Large* and *Medium* segments dominate overall traffic with a share of 78.5.% and 20.5%, respectively, leaving the *Small* segment's traffic at a mere 1%. Our system design is therefore streamlined to optimize efficiency for traffic-heavy tenants.

Efficiency vs. Visibility Delay. As described in Section 4, our data model is designed to make reporting as efficient as possible by grouping PI data along different dimension attributes over small time windows of configurable size. Increasing window size naturally increases the end-to-end processing delay, but also improves the *compression ratio (CR)* as shown in Equation 1:

$$CR = 1 - \frac{|intermediate\ aggregates|}{|PIs|} \tag{1}$$

[4]We exclude tenants with less than 100k PIs in our analysis as they typically do not use our real-time reporting service and do not contribute to overall traffic significantly.

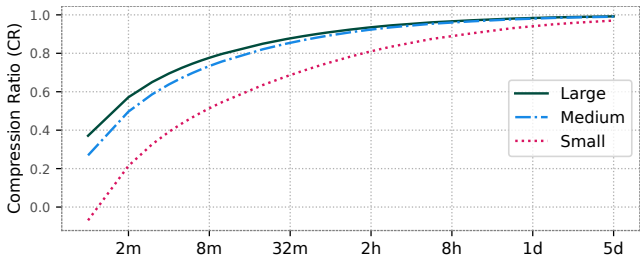

Figure 4: CR by aggregation window size: Increasing window size also increases efficiency, but gains eventually diminish.

The CR is a pivotal metric for performance in our reporting scheme, because it directly represents the efficiency of our dashboarding queries: With a CR of 70%, for example, a query executed over the intermediate aggregates touches 70% fewer records than a query over raw data would. Figure 4 shows the compression ratio (CR) for the three tenant segments under varying aggregation window sizes from 1 minute up to 5 days (logarithmic x-scale). Compared with raw storage, our aggregation scheme with the baseline 1-minute windows provides an efficiency improvement of 37% for *Large* and 27% for *Medium* tenants. In contrast, *Small* tenants have a negative baseline efficiency (-6%) and only benefit from a minimum of 2-minute windows (21%). This is intuitively plausible as the number of PIs in a given timeframe (denominator in Equation 1) is by definition larger for traffic-heavy tenants. Inversely, compression is less efficient for tenants with little traffic per se, simply because fewer PIs are reported at any given time and therefore fewer PIs are written into the same bucket per aggregation window. As another factor for decreased efficiency with *Small* tenants, data collected for a single PI can also be assigned to multiple aggregates when its beacons arrive during the rollover from one intermediate aggregate to the next: This effect may drag down CR for low-traffic tenants as it can lead to the creation of multiple aggregates for single PIs when there are no other PIs falling into the same bucket in a given time window. However, the amount of intermediate aggregates written to our database (numerator in Equation 1) depends not only on the number of PIs and the chosen aggregation window, but also on the dimension attributes. Adding an additional dimension always increases the number of intermediate aggregates written per time interval and thus decreases compression efficiency. The concrete dimension attributes in our evaluation differ for each tenant as they depend on the individual requirements.

Across all tenant segments, increasing aggregation window size can have a dramatic effect on CR especially for small time windows: For example, choosing 2- instead of 1-minute windows improves CR by over 31% for *Large* and *Medium* tenants and still by 26% for *Small* tenants. But efficiency gains eventually diminish with increasing window size: For example, doubling aggregation windows already greater than 1 day only leads to CR improvements below 2%, irrespective of the tenant's traffic numbers. While CR approaches 100% for very large aggregation windows in all segments, traffic-heavy tenants naturally reach high CR values faster: The average *Large* tenant reaches a CR of >80% with a window size of 1 hour, while typical tenants in the *Small* segment would have to tolerate delays upwards of 16 hours to reach the same level of efficiency. This is an important aspect to consider as window size also corresponds to the granularity in which data can be queried. Choosing excessively large aggregation windows is therefore impractical not only due to a huge visibility delay, but also because it translates to low resolution for data analyses. In other words, you cannot analyze daily performance in the dashboard with 1-week aggregation windows.

Summary & Discussion. The analysis confirms that our approach scales well with traffic and offers particularly high efficiency for the *Medium* and *Large* tenants which are responsible for the majority of overall traffic in production. The CR further depends on aggregation window size and can be traded off against the time that data is buffered for aggregation: Better compression thus comes with a certain visibility delay as data can only be queried after the current aggregation window has closed. Our aggregation scheme for real-time reporting is particularly efficient for larger tenants, because more PIs are compressed into the individual intermediate aggregates when there is more traffic to begin with. We thus achieve a good CR for these tenants with 1-minute aggregation windows already. Although increasing window size beyond this baseline still enables significant efficiency gains, further increasing window size (and thus visibility delay) does not provide much benefit when aggregation windows are already large. By processing real-time data in small aggregation windows first (e.g. 1-minute windows) and recompressing it later (e.g. by hour or day), a low visibility delay can be combined with a high CR. However, the incurred processing overhead should be taken into account here as recompression may generate substantial load on the storage system.

6 DEVELOPER EXPERIENCES

In this section, we would like to highlight two rather simple yet pivotal insights we gathered from using Beaconnect in production.

6.1 Alignment With External Tooling is Hard

While Beaconnect is optimized for measuring the effects of page load acceleration via A/B testing, the reporting stack of most Speed Kit customers is not. In consequence, the measurements taken with the customers' existing monitoring are often skewed between test groups: When pages load faster, beacons are transmitted earlier and the chance of beacons getting lost is reduced. But since both test groups load with different speeds, this effect can distort results as certain data points (e.g. early bounces) only become visible for one of the two groups. While accelerating the page load may thus introduce skew by improving data quality for only one of the two test groups, the opposite is also true: Accelerating the navigation from one page to the next can also *increase* the rate of dropped data beacons, because it also reduces the time available for data transmission. This can be an issue, for example, when the actual data endpoint is hidden behind a chain of redirects as the browser only resolves redirects from one page until the HTML of the next page has been fully loaded.

6.2 Continuous Anomaly Detection is Priceless

We have also learned the importance of reliable alerting mechanisms to identify potential issues as early as possible. To minimize response times on technical incidents and proactively point our

customers towards potential issues with their deployments, we are currently developing a component for anomaly detection. In more detail, we are evaluating a prototypical anomaly detection pipeline in our staging deployment as a third processing path next to the ones presented in Section 4. In its current form, the prototype maintains basic metrics like PI or session counters in varying granularity (minutes, hours, days). These summaries are then fed to and analyzed by specialized detector agents to uncover and generate alerts for anomalies such as a sudden increase in bounces, page reloads, or other data artifacts that may be indications of a potential deployment issue.

7 CONCLUSION

Split testing is the de facto standard for evaluating incremental improvements in many areas of Web development. But due to its incompatibility with traditional content delivery networks (CDNs), naturally skewed before-after comparisons have become the norm for measuring the effects of page speed acceleration.

In this paper, we present Beaconnect as a novel approach for Web performance monitoring that enables statistically sound A/B tests. Beaconnect is built for performance monitoring with the Speed Kit technology and is therefore not constrained by the limitations of traditional CDN-based approaches. Our stack has been running in production since 2020 and currently handles over 3 billion data beacons from 650+ million page impressions (PIs) and 100+ million unique users per month. Through a dual aggregation scheme that processes the unsampled performance data, Beaconnect combines subminute end-to-end latency for continuous real-time reporting with support for offline data warehouse analytics. By further querying intermediate aggregates rather than raw beacon data, our real-time reporting approach achieves very high efficiency that can be traded off against a configurable delay for aggregation. We analyze real customer traces to quantify the benefits of our approach and thereby also shed light on the trade-off between query efficiency on the one side and the tunable visibility delay on the other.

ACKNOWLEDGMENTS

We would like to thank the AWS EMEA Prototyping Labs team as well as Daniel Zäh for continuously supporting us in both technical as well as organizational challenges throughout the development of Beaconnect. We also want to thank Carlos Matos from New York City, New York for his inspiring 2017 speech that sparked the idea to name our stack "Beaconnect".

REFERENCES

[1] 2022. Akamai. https://www.akamai.com. (accessed: May 09, 2022).
[2] 2022. Cloudflare. https://www.cloudflare.com/. (accessed: May 09, 2022).
[3] 2022. web.dev. https://web.dev/. (accessed: May 09, 2022).
[4] 2022. WebPageTest. https://www.webpagetest.org/. (accessed: May 09, 2022).
[5] AB Tasty. 2022. The Complete Guide to A/B Testing. https://www.abtasty.com/ab-testing/. (accessed May 08, 2022).
[6] Adobe. 2022. Adobe Analytics. https://www.adobe.com/de/analytics/adobe-analytics.html. (accessed May 08, 2022).
[7] Marcos K. Aguilera, Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen. 2003. Performance Debugging for Distributed Systems of Black Boxes. *ACM SIGOPS Review* 37, 5 (2003), 74–89.
[8] Akamai. 2022. mPulse. https://www.akamai.com/uk/en/products/performance/mpulse-real-user-monitoring.jsp. (accessed: May 09, 2022).
[9] Alex Russell and Jungkee Song and Jake Archibald and Marijn Kruisselbrink. 2019. Service Workers 1: W3C Candidate Recommendation. https://www.w3.org/TR/service-workers/. (accessed May 08, 2022).
[10] Amazon. 2022. Amazon CloudFront. https://aws.amazon.com/cloudfront/. (accessed: May 09, 2022).
[11] AWS. 2022. Amazon Kinesis Data Streams. https://aws.amazon.com/kinesis/data-streams/. (2022). (accessed May 08, 2022).
[12] AWS. 2022. Amazon S3. https://aws.amazon.com/s3/. (2022). (accessed May 08, 2022).
[13] Benjamin Wollmer and Wolfram Wingerath. 2020. Mobile Site Speed – The User Perspective. https://medium.baqend.com/16cd77f9ce25. *Baqend Tech Blog* (2020). (accessed May 08, 2022).
[14] Tim Berners-Lee, Roy T. Fielding, and Henrik Frystyk Nielsen. 1996. *Hypertext Transfer Protocol – HTTP/1.0.* RFC 1945. RFC Editor. http://www.rfc-editor.org/rfc/rfc1945.txt (accessed November 22, 2021).
[15] Leszek Borzemski. 2010. The Experimental Design for Data Mining to Discover Web Performance Issues in a Wide Area Network. *Cybernetics and Systems: An International Journal* 41, 1 (2010), 31–45.
[16] Leszek Borzemski, Marta Kliber, and Ziemowit Nowak. 2009. Using Data Mining Algorithms in Web Performance Prediction. *Cybernetics and Systems: An International Journal* 40, 2 (2009), 176–187.
[17] Anna Bouch, Allan Kuchinsky, and Nina Bhatti. 2000. Quality is in the Eye of the Beholder: Meeting Users' Requirements for Internet Quality of Service. In *ACM SIGCHI*. 297–304.
[18] Jin Cao, William S. Cleveland, Yuan Gao, Kevin Jeffay, F. Donelson Smith, and Michele Weigle. 2004. Stochastic Models for Generating Synthetic HTTP Source Traffic. In *IEEE INFOCOM*, Vol. 3. 1546–1557.
[19] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. 2002. Pinpoint: Problem Determination in Large, Dynamic Internet Services. In *DSN*. 595–604.
[20] Jürgen Cito, Devan Gotowka, Philipp Leitner, Ryan Pelette, Dritan Suljoti, and Schahram Dustdar. 2015. Identifying Web Performance Degradations Through Synthetic and Real-User Monitoring. *Journal of Web Engineering* (2015), 414–442.
[21] Dynatrace. 2022. Dynatrace Real-User Monitoring (RUM). https://www.dynatrace.com/platform/real-user-monitoring/. (accessed May 08, 2022).
[22] Elasticsearch. 2022. Elasticsearch. https://www.elastic.co/. (2022). (accessed May 08, 2022).
[23] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. 2017. The Benefits of Controlled Experimentation at Scale. In *SEAA*. 18–26.
[24] Fastly. 2022. Fastly. https://www.fastly.com/. (accessed May 08, 2022).
[25] Felix Gessert and Wolfram Wingerath. 2020. Batching was Yesterday! Real-Time Tracking For 100+ Million Visitors. https://emea-resources.awscloud.com/dach-events-webinars/batching-was-yesterday-real-time-tracking-for-100-million-visitors. *AWS Webinar* (2020). (accessed May 08, 2022).
[26] Ellen Friedman and Kostas Tzoumas. 2016. *Introduction to Apache Flink: Stream Processing for Real Time and Beyond* (1st ed.). O'Reilly Media, Inc.
[27] Google. 2021. How a Web Session is Defined in Universal Analytics. https://support.google.com/analytics/answer/2731565. (accessed November 11, 2021).
[28] Google. 2022. About Data Sampling. https://support.google.com/analytics/answer/2637192. (accessed May 08, 2022).
[29] Google. 2022. Data Limits for Universal Analytics Properties. https://support.google.com/analytics/answer/1070983. (accessed May 08, 2022).
[30] Google. 2022. Google Analytics. https://analytics.google.com. (accessed: May 09, 2022).
[31] Kibana. 2022. Kibana. https://www.elastic.co/kibana. (2022). (accessed May 08, 2022).
[32] Andrew B. King. 2003. *Speed up Your Site: Web Site Optimization.* New Riders.
[33] Ron Kohavi, Randal M. Henne, and Dan Sommerfield. 2007. Practical Guide to Controlled Experiments on the Web: Listen to Your Customers, Not to the Hippo. In *ACM SIGKDD*. ACM, 959–967.
[34] Ron Kohavi and Roger Longbotham. 2017. Online Controlled Experiments and A/B Testing. *Encyclopedia of Machine Learning and Data Mining* 7, 8 (2017), 922–929.
[35] George Pallis and Athena Vakali. 2006. Insight and perspectives for content delivery networks. *Commun. ACM* 49, 1 (2006), 101–106.
[36] Al-Mukaddim Khan Pathan and Rajkumar Buyya. 2007. A Taxonomy and Survey of Content Delivery Networks. *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report* 4 (2007), 70.
[37] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, and Christopher Berner. 2019. Presto: SQL on Everything. In *ICDE*. 1802–1813.
[38] Dan Siroker and Pete Koomen. 2013. *A/B Testing: The Most Powerful Way to Turn Clicks Into Customers.* John Wiley & Sons.
[39] Swaminathan Sivasubramanian. 2012. Amazon DynamoDB: A Seamlessly Scalable Non-Relational Database Service. In *SIGMOD*. 729–730.
[40] Wolfram Wingerath. 2019. Big Data Analytics With AWS Athena. https://youtu.be/POUrpC8hqWU. *Code Talks* (2019). (accessed May 08, 2022).
[41] Wolfram Wingerath, Felix Gessert, Erik Witt, Hannes Kuhlmann, Florian Bücklers, Benjamin Wollmer, and Norbert Ritter. 2020. Speed Kit: A Polyglot & GDPR-Compliant Approach For Caching Personalized Content. In *ICDE*. 1603–1608.

[42] Wolfram Wingerath, Norbert Ritter, and Felix Gessert. 2019. *Real-Time & Stream Data Management: Push-Based Data in Research & Practice.* Springer International Publishing.

[43] Wolfram Wingerath, Benjamin Wollmer, Markus Bestehorn, Daniel Zaeh, Florian Bücklers, Jörn Domnik, Anil Sener, Stephan Succo, and Virginia Amberg. 2021. How Baqend Built a Real-Time Web Analytics Platform Using Amazon Kinesis Data Analytics for Apache Flink. https://aws.amazon.com/de/blogs/big-data/how-baqend-built-a-real-time-web-analytics-platform-using-amazon-kinesis-data-analytics-for-apache-flink/. *AWS Big Data Blog* (2021). (accessed May 09, 2022).

[44] Wolfram Wingerath and Benjamin Wollmer. 2020. Mobile Site Speed – The Business Perspective. https://medium.baqend.com/77c5852e2743. *Baqend Tech Blog* (2020). (accessed May 08, 2022).

[45] Xiaoqian Wu. 2020. A Primer for Web Performance Timing APIs. https://w3c.github.io/perf-timing-primer/. *W3C Editor's Draft* (2020). (accessed February 25, 2021).

[46] Scott W. H. Young. 2014. Improving Library User Experience with A/B Testing: Principles and Process. *Weave: Journal of Library User Experience* 1, 1 (2014). https://doi.org/10.3998/weave.12535642.0001.101

[47] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *USENIX HotCloud*. 10.

[48] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *ACM SOSP*. Association for Computing Machinery, 423–438.