The Anatomy of a Large-Scale Online Experimentation Platform

Somit Gupta, Lucy Ulanova, Sumit Bhardwaj, Pavel Dmitriev, Paul Raff
Analysis and Experimentation Team, Microsoft
Bellevue, WA, USA
sogupta,liulanov,subhardw,padmitri,paraff@microsoft.com

Aleksander Fabijan
Department of Computer Science and Media Technology
Malmö University
Malmö, Sweden
aleksander.fabijan@mau.se

Abstract— **Online controlled experiments (e.g., A/B tests) are an integral part of successful data-driven companies. At Microsoft, supporting experimentation poses a unique challenge due to the wide variety of products being developed, along with the fact that experimentation capabilities had to be added to existing, mature products with codebases that go back decades. This paper describes the Microsoft ExP Platform (ExP for short) which enables trustworthy A/B experimentation at scale for products across Microsoft, from web properties (such as bing.com) to mobile apps to device drivers within the Windows operating system. The two core tenets of the platform are *trustworthiness* (an experiment is meaningful only if its results can be trusted) and *scalability* (we aspire to expose every single change in any product through an A/B experiment). Currently, over ten thousand experiments are run annually. In this paper, we describe the four core components of an A/B experimentation system: experimentation portal, experiment execution service, log processing service and analysis service, and explain the reasoning behind the design choices made. These four components work together to provide a system where ideas can turn into experiments within minutes and those experiments can provide initial trustworthy results within hours.**

Keywords— A/B testing; experimentation; metrics; hypothesis testing; experimentation platform; scalability.

I. INTRODUCTION

Online controlled experiments (e.g., A/B tests) are becoming the gold standard for evaluating improvements in software systems [1]. From front-end user-interface changes to backend algorithms, from search engines (e.g., Google, Bing, Yahoo!) to retailers (e.g., Amazon, eBay, Etsy) to social networking services (e.g., Facebook, LinkedIn, Twitter) to travel services (e.g., Expedia, Airbnb, Booking.com) to many startups, online controlled experiments are now utilized to make data-driven decisions at a wide range of companies [2]. While the theory of a controlled experiment is simple, and dates to Sir Ronald A. Fisher' [3], the deployment of online controlled experiments at scale (for example, 100's of concurrently running experiments) across variety of web sites, mobile apps, and desktop applications is a formidable challenge.

Consider the following experiment that ran in Skype mobile and desktop apps. When the user attempts a call but the caller does not answer, a dialog is shown prompting the user to leave a video message.

Many steps need to happen to execute this experiment. First, the code that shows the message needs to be deployed to the clients in such a way that it can be turned on and off via the experimentation system. Then, the audience, the experiment design, the experiment steps, and the size and duration for each step need to be determined. During the experiment execution, correct configurations need to be delivered to the users' mobile and desktop apps, ideally in a gradual manner, while verifying that the experiment is not causing unintended harm (e.g. app crashes). Such monitoring should continue throughout the experiment, checking for variety of issues, including interactions with other concurrently running experiments. During the experiment, various actions could be suggested to the experiment owner (the person who executes the experiment), such as stopping the experiment if harm is detected, looking at surprising metric movements, or examining a specific user segment that behaves differently from others. After the experiment, results need to be analyzed to make a ship/no-ship recommendation and learn about the impact of the feature on users and business, to inform future experiments.

The role of the experimentation platform in the process described above is critical. The platform needs to support the steps above in a way that allows not only a data scientist, but any engineer or product manager to successfully experiment. The stakes are high. Recommending an incorrect experiment size may result in not being able to achieve statistical significance, therefore wasting release cycle time and resources. Inability to detect harm and alert the experiment owner may result in bad user experience, leading to lost revenue and user abandonment. Inability to detect interactions with other experiments may lead to wrong conclusions.

At Microsoft, the experimentation system (ExP) [4] supports trustworthy experimentation in various products, including Bing, MSN, Cortana, Skype, Office, xBox, Edge, Visual Studio, Windows OS. Thousands use ExP every month to execute and analyze experiments, totaling over ten thousand experiments a year.

In this paper, we describe the architecture of ExP that enables such a large volume of diverse experiments, explain the reasoning behind the design choices made, and discuss the alternatives where possible. While the statistical foundations of online experimentation and their advancement is an active research area [5], [6], and some specific aspects of experiment execution such as enabling overlapping experiments [7] were discussed elsewhere, to our knowledge this is the first work focusing on the end-to-end architecture of a software system supporting diverse experimentation scenarios at scale.

The two core tenets of ExP are *trustworthiness* and *scalability*. "Getting numbers is easy; getting numbers you can trust is hard" [8]. Producing trustworthy results requires solid statistical foundation as well as seamless integration, monitoring, and variety of quality checks of different components of the system. Scalability refers to the ability of a new product or team to easily onboard and start running trustworthy experiments on ExP at low cost, and then enable more and more ExP features as the experimentation volume grows, until every single product feature or bug fix, as long as it is ethical and technically feasible, is evaluated via an experiment [9].

The four core components of ExP are *experimentation portal*, *experiment execution service*, *log processing service*, and *analysis service*. Experimentation portal is an interface between the experiment owner and the experimentation system, enabling the owner to configure, start, monitor and control the experiment throughout its lifecycle (see Figure 1). Experiment execution service is concerned with how different experiment types are executed. Log processing service is responsible for collecting and cooking the log data and executing various analysis jobs. Analysis service is used throughout the whole experiment lifecycle, informing experiment design, determining parameters of experiment execution, and helping experiment owners interpret the results.

The contribution of our paper is threefold:

1. We describe the architecture of ExP, Microsoft's large scale online experimentation platform, supporting experiments across web sites, mobile and desktop apps, gaming consoles, and operating systems.

2. We discuss choices and alternatives, which, in other scenarios and for other companies, may potentially be better options when building their own experimentation platform.

3. We illustrate the discussion with examples of real experiments that were ran at Microsoft.

With increasing use of experimentation in software development, more and more companies are striving to expand their experimentation capabilities. We hope that this paper can serve as a guide, helping them build trustworthy and scalable experimentation solutions, ultimately leading to more data driven decisions benefitting the products and their users.

The rest of the paper is organized as follows. Section II provides a brief overview of online controlled experiments, related work, and our research approach. Section III contains our main contributions, discussing the four core components of ExP in sections III.A-III.D. Section IV concludes the paper.

II. BACKGROUND & RESEARCH METHOD

A. Online Controlled Experiments

In the simplest controlled experiment or A/B test users are randomly assigned to one of the two variants: control (typically denoted as A) or treatment (typically denoted as B). Elsewhere in this paper, we will use V1, V2, V3… to represent variants, and an experiment is defined as a set of variants, e.g. E = {V1, V2}. Usually control is the existing system, and treatment is the existing system with a change X (for example, a new feature). While the experiment is running, user interactions with the system are recorded, and metrics are computed. If the experiment were designed and executed correctly, the only thing consistently different between the two variants is the change X. External factors are distributed evenly between control and treatment and therefore do not impact the results of the experiment. Hence any difference in metrics between the two groups must be due to the change X (or a random chance, that we rule out using statistical testing). This establishes a causal relationship between the change made to the product and changes in user behavior, which is the key reason for widespread use of controlled experiments for evaluating new features in software. For the discussion that follows, an important aspect is where an experiment is executed. If the experiment is executed by the user's mobile/desktop app, gaming console, OS, etc. we label it a *client-side experiment*. If the experiment is executed by the web site, call routing infrastructure, or other server component, we label it a *server-side experiment*.

B. Related Work

Controlled experiments are an active research area, fueled by the growing importance of online experimentation in the software industry. Research has been focused on topics such as new statistical methods to improve metric sensitivity [10], metric design and interpretation [11], [12], projections of results from a short-term experiment to the long term [13], [14], the benefits of experimentation at scale [2], [15], experiments in social networks [16], as well as examples, pitfalls, rules of thumb and lessons learned from running controlled experiments in practical settings [17], [18], [19]. These works provide good context for our paper and highlight the importance of having a solid engineering platform on top of which the new research ideas, such as the ones referenced, can be implemented and evaluated. Our work describes how to build such a platform.

High-level architecture of an experimentation system was discussed in section 4.1 of [20] and partially in [9], [21]. The discussion, however, is very brief, and, most importantly, only concerns experimentation on web properties. Similarly this is the case in [7]. Supporting client experimentation, such as desktop or mobile apps, requires a different architecture. Moreover, since [20] and [7] were published, new experiment design and execution approaches were developed, e.g. [10], which also require a different architecture to support them. Therefore, to introduce a detailed architectural design of an experimentation platform that supports the aforementioned experimentation scenarios, we conducted an in-depth case study at Microsoft, which we briefly describe next.

C. Research Method

The result of this paper builds on case study research [22] conducted at Microsoft Corporation.

Case Company. Microsoft is a multinational organization and employs over 120 000 people. Most of the authors of this paper have been working at the case company for several years with many product teams (e.g. Bing, Skype, Office, XBOX etc.) specifically focusing on enabling and advancing their experimentation capabilities.

Data collection. The authors of this paper have good access to the meeting notes, design documents, notebooks and the experimentation platform itself. We collected and placed all existing documents that describe the ExP in a shared folder, and complemented them with a joint notebook with personal notes and insights about ExP that were previously not documented.

Data Analysis. We aggregated the collected data describing the ExP platform, and jointly triangulated the historical data describing the reasoning behind the various architectural decisions of the ExP platform with our recent and extensive experience. In principal, we applied the thematic coding approach where we grouped similar concepts under categories, and aggregated these further until we agreed on a level of abstraction. In this way, for example, we emerged with "Experiment Execution Service" label and other components that describe our platform. During the analysis process, several people working at the case company reviewed our work and provided feedback on it.

Threats to Validity. *Construct validity:* Our case company has a long tradition of experimentation research and applying it in practice. The authors of this paper and others that contributed to it are very familiar with the architecture of the ExP platform, and the needs and challenges that it has to address. *External Validity:* The discussion in this paper and its main contribution is based on a system implemented at a single company. This is a limitation that we acknowledge. Despite this concern, however, we believe that the diverse range of products that our platform supports make our contributions generalizable to many other scenarios across the software industry, including to companies developing web sites, mobile and desktop apps, gaming consoles, services, and operating systems.

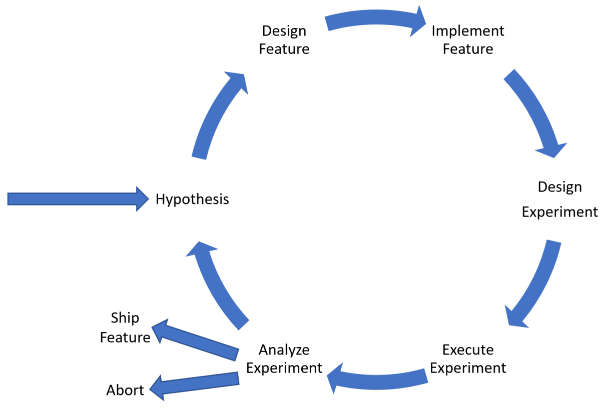


Figure 1. The experimentation lifecycle.

III. EXPERIMENTATION PLATFORM ARCHITECTURE

The ExP platform aims to facilitate the *experimentation lifecycle* (see Figure 1*)*, which is the basis of data-driven product development at Microsoft.

A. *Experimentation Lifecycle*

All product changes (e.g. new features) should start with a *hypothesis*, which helps explain the reasoning for the change. Good hypotheses originate from data analysis and should be quantifiable. In the Skype experiment example introduced in Section 1, the hypothesis could be that *at least 1 out 10 of users who had an unanswered call would leave a video message,* leading to an increase in the number of calls (due to more returned calls) and an increase in the overall use of Skype.

The hypothesis then goes through the steps of implementation and evaluation via an experiment. At the end of the experiment, the decision is made to ship the change or abort. Regardless of the outcome, the learnings from the experiment become the basis of a new hypothesis, either an incremental iteration on the current feature, or something radically different.

In the remainder of this section, we discuss the end-to-end architecture of the experimentation system, which we also depict on Figure 3 on the next page.

B. *Experimentation Portal*

Experimentation portal serves as an interface between the experiment owner and the experimentation system. The goal of the experimentation portal is to make it easy for the experiment owner to create, configure, run, and analyze experiments. For experimentation to scale, we need to keep making these tasks easier.

The three components of the experimentation portal are *experiment management*, *metric definition and validation*, and *result visualization and exploration.*

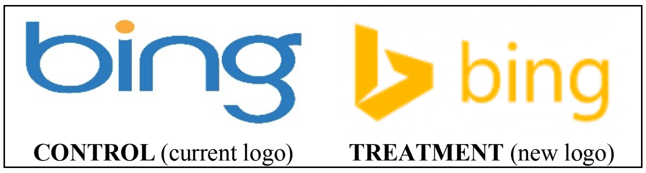


Figure 2. An iteration of the Bing logo experiment.

1) **EXPERIMENT MANAGEMENT**: Consider the following example of an experiment ran in Bing, aimed to understand the impact of changing the Bing's logo, as shown in Figure 2. We will use this example to illustrate the key choices the experiment owner has to make when creating and managing an experiment.

a) **Audience** often depends on the design of the feature. For the example in Figure 2, since this is an English language logo, only users in the United States-English market were exposed to the experiment. It is a common practice to hone your feature and learnings in one audience, before proceeding to expand to other audiences. The experimentation portal needs to provide flexible audience selection mechanisms, also known as *targeting* or *traffic filters*. Examples of simple traffic filters are market, browser, operating system, version of the mobile/desktop app. More complex targeting criteria may involve prior usage information, e.g. "new users" who started using the app within the last month.

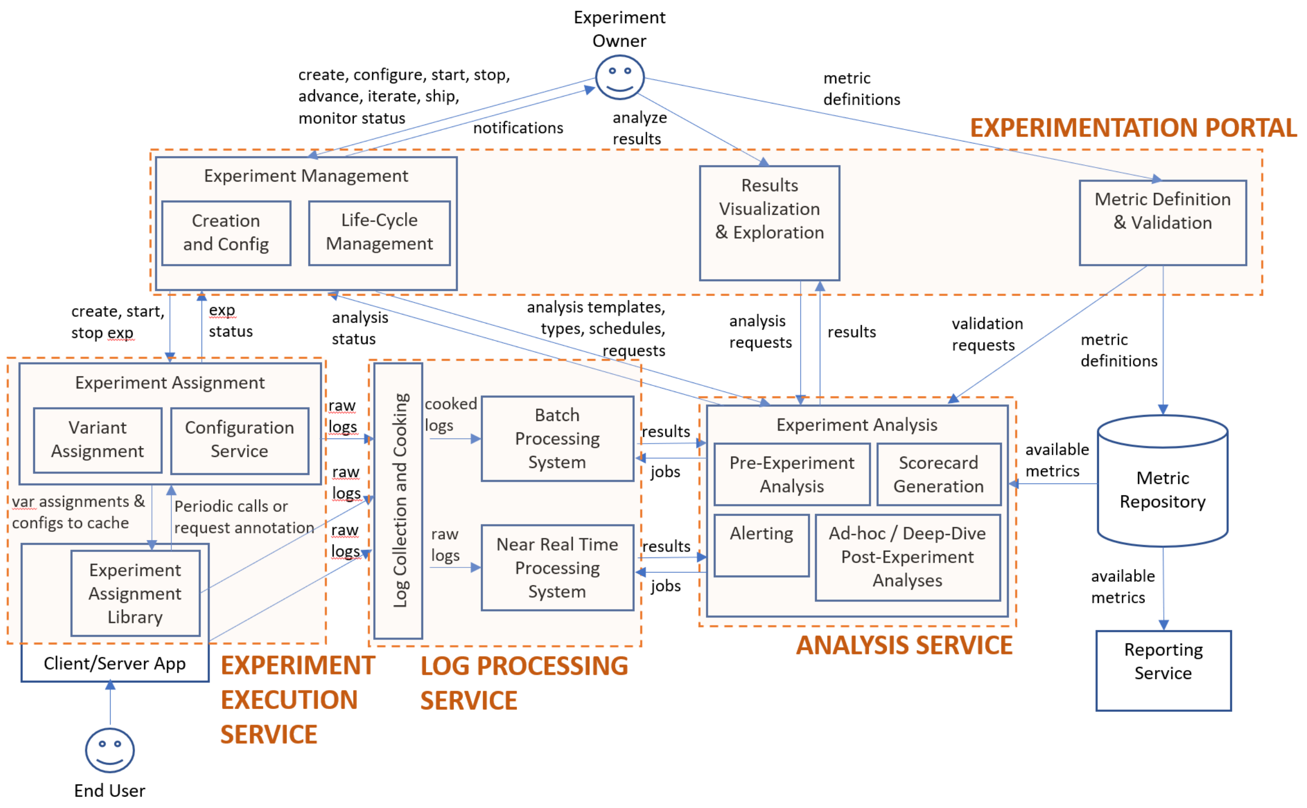


Figure 3. Architecture of the experimentation platform.

b) **Overall Evaluation Criteria (OEC)** is, essentially, the formalization of the success of experiment hypothesis in terms of the expected impact on key product metrics. It is important that the OEC is determined before the experiment starts, to avoid the temptation of tweaking the ship criteria to match the results, and because it serves as input for determining the experiment size and duration. Ideally, most experiments should optimize for a single product-wide OEC rather than individual experiment-specific OECs [9]. For the example in Figure 2, the OEC was the standard Bing OEC discussed in [23]. The experimentation portal should require the experiment owner to formally document the OEC when creating an experiment.

c) **Size and duration.** The experiment owner needs to specify what fraction of the targeted population will receive the treatment and the control, and **how long** the experiment should run. The key tradeoff here is that the experiment owner would be risk averse and would want to exposure as few users as possible to the new feature but still be able to detect sizable changes in key metrics. Power analysis [24] helps in making this decision. Do note that running an experiment longer is not always better in terms of detecting small changes in metrics [23]. One key input in power analysis is the *coverage* of the new feature: the proportion of users in treatment who will experience the new feature. In cases of small coverage triggered analysis can help improve the power of metrics [18]. Other factors like apriori expectaction of novelty or primacy effects in features, or warm up period needed for machine learning models may also be a factor for determining the duration of an experiment. It is important to stick to the duration of the experiment. Stopping the experiment early or increasing the duration after looking at the results can lead to wrong conclusions [19].

d) **Experiment template.** While the basic experiment setup described in Section II.A can be used, it may suffer from *random imbalance*. When the users are randomized into variants, just by chance there may be an imbalance resulting in a statistically significant difference in a metric of interest between the two populations. Disentangling the impact of such random imbalance from the impact of the treatment may be hard. Following the idea of [25], ExP supports several types of experiments that use *re-randomization* to alleviate this problem. The naïve way to detect random imbalance is to start an experiment as an A/A (treatment and control are the same in this stage), run it for several days, and if no imbalance is detected, apply the treatment effect to one of the A's. If random imbalance is present, however, the experiment needs to be restarted with a re-randomization, resulting in a delay. When this experiment type was used in Bing about 1 out of 4 experiments had to be restarted. Another option is to perform a *retrospective A/A analysis* on historical data, the *Pre-Experiment Analysis* step in Figure 3. For example, we can take the historical data for the last one week and simulate the randomization on that data using the hash seed selected for the A/B experiment about to start, as if a real A/A experiment ran during that time, and check for imbalance. Expanding on the above idea, we can perform many (e.g. 1000) retrospective A/A steps simultaneously and then select the most balanced randomization hash seed to use for the A/B step. We call this experiment type *SeedFinder*, referring to its ability to find a good hash seed for randomization. To implement SeedFinder, the experimentation portal needs to interact with the analysis service even before the experiment has started, and the analysis service needs to be able to precisely simulate offline the targeting and randomization as if it happened in the online system. Another aspect of experiment configuration is *ramp-up*: the sequence of steps the experiment will take to reach its final size. Ramp-up is particularly important for risky experiments, where the experiment owner may want to start them at a very small size and gradually move to the final desired size. ExP platform encapsulates different options for experiment types and ramp-up scheduled into experiment templates. Each product has its own set of templates that may include, in addition to the aspects described above, choices for what randomization unit to use in the experiment and alternative experiment designs such as interleaving [26]. Each product has a default recommended template.

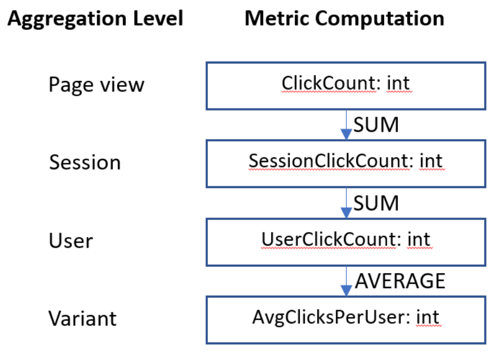
e) **Experiment Interacitons.** When many experiments configured as discussed earlier in this section are running concurrently, every user has a chance to be in any subset of these experiments at the same time. While often it does not create any issues, it is problematic in cases where interactions occur between variants of different experiments. Consider a hypothetical example where one experiment changes the font color on a web page to blue, while another changes the background color to blue. A mechanism to prevent the user from being in both of these experiments at the same time is needed. To address the need to execute multiple potentially interacting experiments concurrently, we use the notion of *isolation group*. Each variant V is tagged with one or more isolation groups, denoted by $IG(V)$. Isolation groups must satisfy the *Isolation Group Principle*: if $V1$ and $V2$ are two variants and $IG(V1) \cap IG(V2) \neq \emptyset$, then a user can never be concurrently assigned to both $V1$ and $V2$. To avoid assignment mismatches between the variants, we require all variants in the experiment to share the same set of isolation groups. The problem of taking a global set of variants – each with its own set of isolation groups – and providing a set of assignments for each user will be discussed in Section III.C.

f) **Variant behavior.** Experiment owner needs to configure the behavior for each variant. For server-side experiments, where the product code can be updated at any time, configuration management system such as the one described in [27] can be used to manage large amount of configurations. For client-side experiments, where the software is only updated periodically, an online configuration system needs to be used to turn the features of the product on and off. Discussion of the design of configuration systems is beyond the scope of this paper.

2) **METRIC DEFINITION AND VALIDATION**: To obtain accurate and comprehensive understanding of experiment results, a rich set of metrics is needed [19]. Therefore, to scale experimentation it is important to provide an easy way to define and validate new metrics, that anyone in the company is able to use. The process of adding new metrics needs to be lightweight, so that it does not introduce delays into the experiment life cycle. It is also important to ensure that metric definitions are consistent across different experiments and business reports.

To satisfy the above requirements, we developed a Metric Definition Language (MDL) that provides a formal programming language-independent way to define all common types of metrics. The user-provided metric definition gets translated into an internal representation, which can then be compiled into a number of target languages, such as SQL. Having a language-independent representation is particularly useful, because different Microsoft products use different backend data processing mechanisms, and sometimes the same product may use several mechanisms. MDL ensures that the definition of a metric is consistent across all such pipelines. See Figure 4 for an example of MDL.

MDL Definition: `AVG(SUM<User>(ClickCount))`



```
PageviewLevel =
SELECT VariantId, UserId, SessionId, ClickCount
FROM Data;

SessionLevel =
SELECT VariantId, UserId, SessionId,
       SUM(ClickCount) AS SessionClickCount
FROM PageviewLevel
GROUP BY VariantId, UserId, SessionId;

UserLevel =
SELECT VariantId, UserId,
       SUM(SessionsClickCount) AS UserClickCount
FROM SessionLevel
GROUP BY VariantId, UserId;

Variant =
SELECT VariantId,
       AVG(UserClickCount) AS AvgClicsPerUser
FROM UserLevel
GROUP BY VariantId;
```

Figure 4. MDL representation of the `AvgClicksPerUser` metric: definition (top), internal representation (middle) and auto-generated SQL (bottom).

Experimentation portal provides a user interface enabling everyone to easily create and edit MDL metric definitions, test them on real data, and deploy to the Metric Repository. This repository is then used by all experiment analysis and business reporting jobs. On average, over 100 metrics are created or edited every day using the metric definition service.

3) **RESULT VISUALIZATION/EXPLORATION**: While the experiment is running, and after its completion, various analysis results are generated: quick near-real time checks are performed, large experiment scorecards with thousands of metrics are computed, alerts are fired, deep-dive analyses are automatically kicked off, and manually configured custom analyses are created. The goal of the Results Visualization and Exploration component is to serve as a single place where experiment analysis information is collected, providing visualizations and tools to help the experiment owner explore the results learning as much as possible, and make the correct ship/no-ship decision. The portal also allows configuring and submitting additional analysis requests.

The simplest form of experiment results is an experiment scorecard – a table consisting of a set of metrics and their movements in an experiment. In a typical scorecard comparing two variants, control and treatment, the following minimal information should be displayed for every metric:

- The observed metric values for control and treatment.
- The delta between these observed values, both in absolute and relative terms.
- The results of a statistical test to convey significance of the delta, typically via a p-value from a t-test.

At Microsoft, it is common for experiment scorecards to compute thousands of metrics over dozens of population segments, making manual scorecard analysis tedious and error prone. Important insights can be missed, especially if they are not expected by the experiment owner or heterogeneity is present in the treatment effect [28], and a large number of metrics gives rise to the multiple comparisons problem [29]. ExP provides a number of visualizations and analysis tools to automatically identify and highlight interesting insights for the experiment owner, and help avoid pitfalls such as the ones mentioned above. Some of these solutions are discussed in [19].

In addition to helping configure, run, and analyze experiments, Experimentation Portal also acts as a central repository of all previously run experiments, collecting metadata about each experiment and its results, and providing anyone an opportunity to learn about the past successes and failures, helping to disseminate learnings from experiments across the company.

C. Experiment Execution Service

The goal of the experiment execution service is generating variant assignments and delivering them to the product.

1) **GENERATING VARIANT ASSIGNMENTS**: At any given point in time, the system has its global experimentation state, which consists of the set of active variants, along with the following information about each variant:

- The set of isolation groups.
- The traffic filters.
- The percentage of total traffic configured.

0	1	3	4	5	6	7	8	9

Figure 5. Diagram of a numberline for a single isolation group. A user who hashes into bucket 7 will always hash into bucket 7.

An assignment request comes with a certain randomization unit (e.g. user id) and as well as other properties that allow resolving the traffic filters. Given this information, the goal of the experiment assignment service is to compute the set of variants for this request.

Let us first consider the scenario of one isolation group. The isolation group has an associated numberline, which can be represented as a sequence of numbers (called buckets), with each bucket representing an equal portion of the population, and the whole representing 100% of the population. For any numberline, each user is consistently hashed into one specific bucket based on their user id, using a hash function such as [30]. For example, Figure 5 shows a numberline with ten buckets, where a user has been assigned to bucket 7.

An experiment that utilizes this isolation group will be allotted a set of buckets, as determined by availability and granularity of the numberline. If an experiment were to utilize this isolation group and consisted of a 40% treatment and a 40% control, then buckets 0-3 could be allotted to treatment and 4-7 to control. In this case, the user who hashes into bucket 7 would be assigned to control for this experiment. This is demonstrated in Figure 6.

Now that assignment for an individual isolation group has been established, we can provide assignment globally by instituting a priority order on isolation groups. Each experiment will utilize the numberline associated with the highest-priority isolation group involved as long as all other isolation groups for that experiment are still available. Then we can iterate through the isolation groups in their priority order, assign on each isolation group, and remove from consideration lower-priority isolation groups that were part of the assigned variant.

2) **DELIVERING VARIANT ASSIGNMENT**: There are three ways in which a client can obtains a treatment assignment, depicted and compared in Figure 7.

a) **Via a service call.** In this simple architecture, the client/server is responsible for explicitly making a service call to the experiment assignment service to obtain treatment assignments. The advantage of this architecture is that it applies equally to sever-side and client-side experiments. The disadvantage is that the service call introduces a delay. Because of this care should be taken to select the appropriate time to make the service call. The typical pattern is to make the call as soon as the application starts (in order to allow experimenting on first run experience), and then periodically, avoiding doing it during important user interactions (e.g. in the middle of a Skype call). Often the application may choose to delay applying the assignment until it restarts, to avoid changing the user experience in the middle of the session.

0	1	2	3	4	5	6	7	8	9
Treatment				Control					

Figure 6. Diagram of a numberline where buckets 0-3 have been allocated to treatment and 4-7 to control.

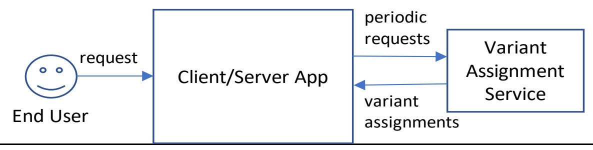
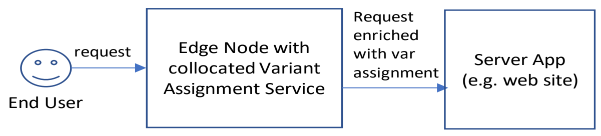
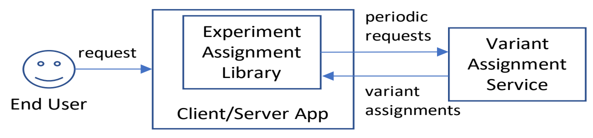
Delivery Type	Architecture	Pros	Cons
Via a service call	 End User request Client/Server App periodic requests variant assignments Variant Assignment Service	Simple and applicable to both server and client experimentation scenarios.	Service call incurs a delay
Via a request annotation	 End User request Edge Node with collocated Variant Assignment Service Request enriched with var assignment Server App (e.g. web site)	No need to make a service request and incur delay penalty.	User request needs to pass through the Edge network, limiting applicability to mostly web sites.
Via a local library	 End User request Experiment Assignment Library Client/Server App periodic requests variant assignments Variant Assignment Service	No need to make a service request. Applicable to both client and server scenarios.	More complex implementation compared to the other approaches.

Figure 7. Comparison of different variant assignment delivery mechanisms.

b) **Via a request annotation.** In this architecture, the service is situated behind an Edge network (the network of servers responsible primarily for load balancing) and treatment assignment is obtained as an enrichment when the user's request passes through the Edge network. The service then obtains the treatment assignment from that enrichement, typically via added HTTP headers. This is a common architecture for web sites because it does not require making an explicit service call. This architecture, however, is not applicable to client experimentation scenarios, because user interaction with clients are often local not requiring remote requests (e.g. editing a document on a PC.)

c) **Via a local library**. In this architecture, the client utilizes solely local resources for treatment assignment. The local library relies on configuration that is periodically refreshed by making a call to the central variant assignment service. This architecture is the best for client experimentation scenarios, and applicable to server experimentation scenarios as well. An application does not need to deal with making service calls explicitly, and also does not need to pass the treatment assignments around the stack as every component in the stack can call the library as often as needed without incurring the delay penalty.

3) **CONFIGURATION**: In addition to obtaining its variant assignment, the client also needs to know how to change its behavior based on these assignments. For server-side experiments, where the code can be updated any time, these variant configurations can be shipped as part of the code. For client-side experiments, updating the code may be a lengthy process, and a typical approach is to ship features dark (turned-off), and then utilize a configuration system to turn them on and off in response to different variant assignments. While variant assignment and configuration services have different goals, it is convenient to collocate them so that configurations are delivered to clients along with variant assignments.

D. Log Processing Service

The job of the log processing service is to collect the different types of log data (e.g. unique identifiers, timestaps, performance signals, context flags, etc.) and then merge, enrich, and join it to make it easier to consume by analysis jobs, the processed we call cooking. The data then becomes available for consumption by different data processing systems. Near real time processing system typically operates on small amounts of recent raw or lightly cooked data and is used to perform time-sensitive computations such as real-time alerts. Batch processing system operates on large volume of data and is used to perform expensive operations, such as computing a scorecard with thousands of metrics over the whole experiment period.

1) **DATA COLLECTION**: From experiment analysis point of view, the goal of data collection is to be able to accurately reconstruct user behavior. During experiment analysis, we will both measure different user behavior patterns, and interpret them as successes and failures, to determine whether the change improves or degrades user experience. In data driven products that make extensive use of experimentation, logging the right data is an integral part of development process, and the code that does it adheres to the same standards (design, unit testing, etc.) as the code that implements core product functionality.

An important type of telemetry data is counterfactual logs. Having such signal allows for triggered analysis, greatly improving the sensitivity of analysis [18].

Telemetry data is usually collected from both the client and the server. Whenever there is a choice of collecting the same type of data from the client or from the server, it is preferable to collect server-side data. Server-side telemetry is easier to update, and the data is usually more complete and has less delay.

Client logs can be lossy – for instance if a web page takes too long to load then the user may close the browser or navigate away before the page loads and client cannot send a data point for it. Clients can also lose data if they are offline for a long time and their cache fills up. It is common for client data to arrive late (some mobile apps only send telemetry when the user is on Wi-Fi). It is important to measure the rate of client data loss in each variant, as it is possible for the treatment variant to affect the data loss rate, potentially leading to incorrect metric calculations [23].

2) **DATA COOKING**: The goal of data cooking is to merge and transform the raw logs from all the different sources into the form that is easy and efficient to consume by analysis jobs. For example, when a web page is shown to the user there's a record about it in the server logs. When the user clicks on a link on the page, there's a record for it in the client logs. During cooking these two records will be merged making it much easier to compute metrics such as click-through rate.

Since telemetry keeps evolving over time, the cooking pipeline needs to be designed in a way that easily accommodates new telemetry events and changes to the existing events. Ideally, for most common telemetry changes, no manual changes to the pipeline should be needed.

An important property of the cooking pipeline is the "No data left behind" principle. It means that any event or event property that existed in the raw logs should, in one form or another, be present in the cooked log. This property is essential both for ensuring the accuracy of analysis results, and for deep dive investigations of the reasons for metric movements.

As discussed in section 4.3.1, data delay and loss are common in client experimentation, and it is the job of the cooking pipeline to handle correctly delayed and incomplete data, update the cooked data as delayed events come in, and enrich the cooked data with extra signals for measuring the rate of data loss.

3) **DATA QUALITY MONITORING**: A critical piece of log processing service is data quality monitoring. Our experience is that intentional and unintentional changes to telemetry and cooking process happen all the time, impacting the correctness of the experimentation metrics. If such changes go unnoticed by experiment owners, metrics may be computed incorrectly, potentially resulting in wrong decisions on experiments.

Typical properties of the data to monitor are volume, fractions of outlier, empty, and null values, and various invariants – the statements about the data that we expect to always hold. An example of an invariant is "every Skype call with the status `Connected=true` should have a non-zero duration", and conversely "every Skype call with the status `Connected=false` should have zero duration". Once the cooked data stream is produced, a separate job needs to run to compute various properties of the stream, monitoring conditions should be evaluated, and alerts triggered if any issues are detected.

E. Experiment Analysis

1) **ANALYSIS LIFECYCLE**: Experiment analysis is conducted during the entire lifecycle of an experiment – hypothesis generation, experiment design, experiment execution, and post experiment during the decision-making process.

a) **Hypothesis Generation**. Results of historical experiments inform new hypotheses, help estimate how likely the new hypothesis is to impact the OEC, and help prioritize existing ideas. During this stage, the experiment owner examines other historical experiments similar to the one he/she is planning to run, as well as historical experiments that improved the metrics he/she is hoping to improve.

b) **Experiment Design**. During experiment design, analysis is performed to answer the following key questions:

- What kind of randomization scheme to use?
- How long should the experiment run?
- What percentage of traffic should be allotted?
- What randomization seed to use to minimize the imbalance?

c) **Experiment Execution**. While the experiment is running, the analysis must answer two key questions:

- Is the experiment causing an unacceptable harm to users?
- Are there any data quality issues yielding untrustworthy experiment results?

d) **Post-Experiment**. At the end of the experiment, to decide on the next steps, analysis should answer the following questions:

- Is the data from the experiment trustworthy?
- Did the treatment perform better than control?
- Did the treatment cause unacceptable harm to other key product metrics?
- Why did the treatment do better/worse than control?
- Are there any heterogeneous effects, with treatment impacting different populations differently?

To answer these and other analysis questions analysis service runs a number of automated and manual analysis jobs, generates notifications and alerts for the experiment owners, and provides tools for deep dive analysis and debugging. We discuss several key analysis topics in more detail next.

2) **METRIC COMPUTATION**: Care needs to be taken to compute the variance correctly during the metric computation. While the variance of the count and sum metrics computed at the randomization unit level is straightforward, percentile metrics and metrics computed at other levels where units are not independent (e.g. average Page Load Time computed as sum of page load times over the number of pages loaded across all users) require the use of delta method and bootstrap [5]. Wrong variance calculation may lead to incorrect p-values and potentially incorrect conclusions about the experiment outcome. In [10] a technique was introduced that uses pre-experiment data to reduce the variance in experimentation metrics. We found that this technique is very useful in practice and is well worth the extra computation time it requires, because in large products very small changes in the metric can have millions of dollars impact on annual revenue.

Another challenge with metric computation is computing large number of metrics efficiently at scale. With thousands of scorecards computed every day, each often requiring to process terabytes of data and perform sophisticated computations such as the one mentioned above, this is a non-trivial challenge. We have found that caching computations that are common across multiple experiments is helpful for reducing data size and performance.

3) **SCORECARD ORGANIZATION**: While the naïve view of the scorecard as an arbitrary organized set of metrics may work for products that are just starting to run experiments, the number of metrics grows quickly, with most products using hundreds to thousands of metrics. This makes examining the metrics table manually very time consuming. Several scorecard organization principles and analysis components can help make it easier for users to understand the results and highlight important information they should not miss.

Metric	T	C	Delta (%)	*p*-value
Overall Click Rate	0.9206	0.9219	-0.14%	8e-11
Web Results	0.5743	0.5800	-0.98%	~0
Answers	0.1913	0.1901	+0.63%	5e-24
Image	0.0262	0.0261	+0.38%	0.1112
Video	0.0280	0.0278	+0.72%	0.0004
News	0.0190	0.0190	+0.10%	0.8244
Related Search	0.0211	0.0207	+1.93%	7e-26
Pagination	0.0226	0.0227	-0.44%	0.0114
Other	0.0518	0.0515	+0.58%	0.0048

Figure 8. Example set of metrics between a treatment (T) and control (C). A full breakdown is shown, indicating and explaining the overall movement.

Another important aspect of scorecard organization is providing breakdowns of key metrics. For example, Figure 8 shows that a success metric Overall Page Clicks Rate has moved negatively in the treatment group. Such movement is much easier to interpret using the breakdown shown below, which shows that the negative movement happened in "Web Results", and that its other components were in fact positive.

4) **VERIFYING DATA QUALITY**: Data quality checks are the first step in analyzing experiment results [31]. One of the most effective data quality checks for experiment analysis is the Sample Ratio Mismatch (SRM) test, which utilizes the Chi-Squared Test to compare the ratio of the observed user counts in the variants against the configured ratio. When an SRM is detected, the results are deemed invalid and the experimenter is blocked from viewing the results to prevent incorrect conclusions.

Another useful data quality verification mechanism is running A/A experiments. In an A/A experiment there is no treatment effect, so if all the computations are correct, the p-values are expected to be distributed uniformly. If p-values are not uniform, it indicates an issue, for example an incorrect variance computation.

Given that data quality issues put in question the validity of OEC and other key experiment metrics, the platform needs to guarantee that these metrics are easily discoverable on the scorecard, and experiment owners are alerted of any degradations.

5) **ALERTING**: In the early stages of growing experimentation in a product, when only a few experiments are run, alerting is an optional feature of the platform as every experiment is carefully designed and monitored by the experiment owner. As experimentation scales, however, there is less scrutiny about every proposed experiment, and less monitoring. Experimentation platform needs to become more intelligent, automatically analyzing experiment results and alerting experiment owners, providing an important layer of live-site security.

Experimentation portal needs to provide functionality to configure alerts for every product running experiments. Typically alerts would be configured for data quality metrics, OEC metrics, performance metrics such as page load time, errors and crashes, and a few other important product characteristics.

To make alerts actionable, statistical analysis needs to be done to determine when to trigger an alert. Note that the goal of analysis here is different from that of understanding experiment results – alerts should only notify experiment owners of serious issues that may require experiment shutdown. It is important that alerts take into account not only the level of statistical significance, but also the effect size. For example, an experimentation platform should not alert on a 1-millisecond degradation to page load time metric, even if the confidence that the degradation is a real effect is high (e.g., the p-value is less than 1e-10.)

Based on the severity of the impact, the system can simply notify the experiment owner, schedule experiment termination and let experiment owner cancel it, or in most severe cases it may terminate the experiment. These configurations need to balance live-site security with allowing experiment owners to be the final authority on shutting down an experiment.

6) **DEEP DIVE ANALYSIS**: While the overall scorecard provides the average treatment effect on a set of metrics, experiment owners should also have an ability to better understand metric movements. Experiment scorecard should provide an option to investigate the data by population segment (e.g. browser type) or time-period (e.g. daily/weekly) and evaluate the impact on each segment. Using this breakdown, experiment owners might discover heterogeneous effects in various segments (e.g. an old browser version might be causing a high number of errors) and detect novelty effects (e.g. the treatment group that received a new feature engages with it in the first day and stops using it after that). This type of analysis, however, needs to be done with care, as it is subject to the multiple comparisons issue. Automatically and reliably identifying such heterogeneous movements is an active research area. ExP platform employs one such method [28], notifying experiment owners if interesting or unusual movements are detected in any segment.

To deeper understand the metric change in a specific segment, the system should provide tools to easily obtain examples of user sessions, containing the events that are used to compute the metric of interest. For example, in an increase in JavaScript errors is observed on a Bing search results page, it may be useful for experimenter to see examples of the error types and queries they triggered on, and conduct simple analysis to see if a specific error type, or a specific query, is causing the increase. While such analyses are conceptually simple, implementing tools and automation to support them is important. Doing such deep-dive analyses manually requires deep understanding of how the data is structured, how a specific metric is computed, and how experiment information is stored in the data, which is beyond the level of knowledge of a typical experiment owner.

IV. CONCLUSION

Online controlled experiments are becoming the standard operating procedure in data-driven companies [2], [20]. However, for these companies to experiment with high velocity and trust in experiment results, having an experimentation platform is critical [9]. In this paper, we described the architecture of a large scale online experimentation platform. Such platform is used by thousands of users at Microsoft, supporting hundreds of concurrent experiments, totally over 10000 experiments per year. This platform enables trustworthy experimentation at scale, adding hundreds of millions of dollars of additional yearly revenue for Microsoft. We hope that the architectural guidance provided in this paper will help companies implement their experimentation systems in an easier, more cost-effective way, helping to further grow the experimentation practices in software industry.

REFERENCES

[1] R. Kohavi and R. Longbotham, "Online Controlled Experiments and A/B Tests," *Encycl. Mach. Learn. Data Min.*, no. Ries 2011, pp. 1–11, 2015.

[2] R. Kohavi and S. Thomke, "The Surprising Power of Online Experiments," *Harvard Business Review*, no. October, 2017.

[3] J. F. Box, "R.A. Fisher and the Design of Experiments, 1922–1926," *Am. Stat.*, vol. 34, no. 1, pp. 1–7, Feb. 1980.

[4] ExP, "Microsoft Experimentation Platform." .

[5] A. Deng, J. Lu, and J. Litz, "Trustworthy Analysis of Online A/B Tests: Pitfalls, Challenges and Solutions," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017, pp. 641–649.

[6] A. Deng, J. Lu, and S. Chen, "Continuous monitoring of A/B tests without pain: Optional stopping in Bayesian testing," in *Proceedings - 3rd IEEE International Conference on Data Science and Advanced Analytics, DSAA 2016*, 2016, pp. 243–252.

[7] D. Tang, A. Agarwal, D. O'Brien, and M. Meyer, "Overlapping experiment infrastructure," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '10*, 2010, p. 17.

[8] R. Kohavi and R. Longbotham, "Online experiments: Lessons learned," *Computer (Long. Beach. Calif).*, vol. 40, no. 9, pp. 103–105, 2007.

[9] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The Evolution of Continuous Experimentation in Software Product Development: From Data to a Data-Driven Organization at Scale," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 770–780.

[10]A. Deng, Y. Xu, R. Kohavi, and T. Walker, "Improving the sensitivity of online controlled experiments by utilizing pre-experiment data," in *Proceedings of the sixth ACM international conference on Web search and data mining - WSDM '13*, 2013, p. 123.

[11]P. Dmitriev and X. Wu, "Measuring Metrics," *Proc. 25th ACM Int. Conf. Inf. Knowl. Manag. - CIKM '16*, pp. 429–437, 2016.

[12]A. Deng and X. Shi, "Data-Driven Metric Development for Online Controlled Experiments," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016, pp. 77–86.

[13]P. Dmitriev, B. Frasca, S. Gupta, R. Kohavi, and G. Vaz, "Pitfalls of long-term online controlled experiments," in *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, 2016, pp. 1367–1376.

[14]H. Hohnhold, D. O'Brien, and D. Tang, "Focusing on the Long-term," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, 2015, pp. 1849–1858.

[15]A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The Benefits of Controlled Experimentation at Scale," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2017, pp. 18–26.

[16]Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin, "From Infrastructure to Culture: A/B Testing Challenges in Large Scale Social Networks," *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, no. Figure 1, pp. 2227–2236, 2015.

[17]R. Kohavi, A. Deng, R. Longbotham, and Y. Xu, "Seven rules of thumb for web site experimenters," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014, pp. 1857–1866.

[18]R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: Survey and practical guide," *Data Min. Knowl. Discov.*, vol. 18, no. 1, pp. 140–181, 2009.

[19]P. Dmitriev, S. Gupta, K. Dong Woo, and G. Vaz, "A Dirty Dozen: Twelve Common Metric Interpretation Pitfalls in Online Controlled Experiments," *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '17*, pp. 1427–1436, 2017.

[20]R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann, "Online controlled experiments at large scale," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, 2013, p. 1168.

[21]R. L. Kaufman, J. Pitchforth, and L. Vermeer, "Democratizing online controlled experiments at Booking. com," *arXiv Prepr. arXiv1710.08217*, pp. 1–7, 2017.

[22]P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2008.

[23]R. Kohavi, A. Deng, B. Frasca, R. Longbotham, T. Walker, and Y. Xu, "Trustworthy Online Controlled Experiments : Five Puzzling Outcomes Explained," *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discov. data Min.*, pp. 786–794, 2012.

[24]R. B. Bausell and Y.-F. Li, *Power analysis for experimental research: a practical guide for the biological, medical and social sciences*. Cambridge University Press, 2002.

[25]K. L. Morgan and D. B. Rubin, "Rerandomization to improve covariate balance in experiments," *Ann. Stat.*, vol. 40, no. 2, pp. 1263–1282, 2012.

[26]F. Radlinski and N. Craswell, "Optimized interleaving for online retrieval evaluation," in *Proceedings of the sixth ACM international conference on Web search and data mining - WSDM '13*, 2013, p. 245.

[27]R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 232–282, Jun. 1998.

[28]A. Deng, P. Zhang, S. Chen, D. W. Kim, and J. Lu, "Concise Summarization of Heterogeneous Treatment Effect Using Total Variation Regularized Regression," *Submiss.*

[29]"Multiple Comparisons Problem," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Multiple_comparisons_problem.

[30]P. K. Pearson, "Fast hashing of variable-length text strings," *Commun. ACM*, vol. 33, no. 6, pp. 677–680, Jun. 1990.

[31]P. Dmitriev, A. Deng, R. Kohavi, and P. Raff, "A / B Testing at Scale : Accelerating Software Innovation," in *SIGIR*, 2017, pp. 1395–1397.