variant

Application Iteration Management as Middleware

April 2019

1. Background

Interactive computer applications are the mainstay of the Internet economy, supporting trillions of dollars of economic activity annually. Most people interact with websites, mobile apps, computer games, IVR systems, etc. on a daily basis. Across all such applications, there are at least two compelling use cases, calling for co-existence of parallel user experiences: *feature toggles* (a.k.a. *feature flags*) and *experiments* (a.k.a. *A/B tests*.)

Feature toggles are employed for gradual and controlled roll-out of new features: users are targeted to the candidate experience gradually and according to some attributes, such as the ZIP code. Feature toggles also provide the mechanism to programmatically shut off all traffic to the candidate experience(s) if a defect is discovered.

In an experiment, the existing code path serves as the control experience and the candidate experience(s) as the treatment(s). User traffic is split randomly between them. The experiment is run until it reaches *statistical significance* or is called off, if the observed difference is too small.

Collectively, feature toggles and experiments are critical tools in managing an application's iteration process. Independent deployment units can be deployed as soon as they are ready, and product ideas can be experimented upon and refined continually. However, instrumentation of such code variations at scale presents novel challenges, as user sessions may be traversing a large number of code variations; targeted experiences must be kept consistent over time, and a viable mechanism must be provided for integration with the host environment.

2. State-of-the-Art Doesn't Work at Scale

All encumbent commercial A/B testing and feature management tools suffer from the same set of basic flaws:

- No unified, dynamic and source controllable metadata dictionary. All changes to the variation metadata must be done manually via a Web console or via a low-level Web API, making the dev ops integration untenable.
- To avoid prohibitively expensive server-to-server calls, server side SDKs work off a replica of the remote variation metadata, presenting multiple synchronization-related problems.
- Server-side SDKs exert unpredictable memory and processor overhead on the host application, jeopardizing host application's core functionality.

- No session management. This limitation has many negative implications and is particularly severe in the case of distributed host applications.
- Integration with the application data must be accomplished in the client code. This creates massive amounts of *instrumentation code smell*, which is not reusable and will have to be cleaned up.
- No support for concurrent code variations.
- Not extensible.

These flaws may not be fatal for smaller, slow evolving applications, but enterprise-scale applications, serving non-trivial user traffic and aiming to conduct multiple experiments at any given time and multiple code deployments on any given day, require a novel approach.

3. Variant's Solution

At the center of Variant's disruptive leap forward are these two principal advances:

- The paradigm-shifting Code Variation Model (CVM) enables unprecedented power to abstract all instrumentation related concerns out of the host application and into an external metadata dictionary called the variation schema.
- Middleware approach, based on the client-server architecture, enables many critical new features, like distributed session management, extensibility, and low network latency.

Code Variation Model provides the abstractions and the grammar for describing a set of code variations, instrumented on a host application. These definitions are grouped together in variation schema files, which are external of the host application and managed by Variant server. Variation schemata enable clean separation between implementation and instrumentation — the application code remains concerned with the implementation of user experiences, while the instrumentation details are handled by Variant.

For example, CVM offers complete support for concurrent variations, which arise frequently in practice because of the Pareto principle — 80 percent of users' time is spent in 20 percent of the application's code. These hot components are typically the subject of multiple experiments and feature rollouts at any given time. Variant's generalized approach to variation concurrency allows application developers, working on different features, to remain autonomous.

Middleware integration point means that Variant server runs alongside the host application, behind the customer's firewall, which entails a number of critical improvements over the incumbent solutions. Thus, Variant server acts as the distributed session manager: Variant sessions are independent of the host application's native sessions, leaving the application programmer free to manipulate the native sessions without the risk of impacting code variations. Variant sessions are particularly useful in a distributed application environment, where multiple cooperating components participate in an experience. Variant sessions also provide the logical isolation context: each session sees a consistent snapshot of the metadata, as it existed at the session's creation time.

Variant sessions also enable unprecedented support for *qualification and targeting persistence*. Application developer may choose between three persistence levels for both qualification and targeting: *unstable*, *stable* and *durable*. By default, both qualification and targeting are stable, i.e. have the life span of a Variant session. However, to guarantee that a return user sees the same experience as before, application programmer can set targeting persistence to durable. In fact, Variant is the only experience variation platform capable of instrumenting a multi-device experiment where a user may start a flow (e.g. a credit card application) on her office desktop, continue on a mobile phone on the train ride home, and finish on a tablet in bed.

Finally, Variant server is highly extensible and integrates organically with the host application's operational data via its server-side Extension API. User extensions are integrated via a call-back mechanism, where Variation server raises various types of lifecycle events and delegates to the extension classes subscribed to these events. For example, a custom qualification algorithm can be supplied in a hook subscribed to the session qualification lifecycle event. Hooks provide a highly portable way for code reuse from one code variation to the next.