# JACKPOT: Online Experimentation of Cloud Microservices

Mert Toslali
*Boston University*

Srinivasan Parthasarathy
*IBM Research*

Fabio Oliveira
*IBM Research*

Ayse K. Coskun
*Boston University*

## Abstract

Online experimentation is an agile software development practice, which plays a central role in enabling rapid innovation. It helps shorten code delivery cycles, which is critical for companies to survive in a competitive software-driven market. Recent advances in cloud computing, including the maturity of container-based technologies and cloud infrastructure, as well as the advent of service meshes, have created an opportunity to broaden the scope of online experimentation and further increase developers' agility.

In this paper, we propose a novel formulation for online experimentation of cloud applications which generalizes traditional approaches applied to web and mobile applications by incorporating the unique challenges posed by cloud environments. To enable practitioners to apply our formulation, we develop and present JACKPOT, a system for online cloud experimentation in the presence of multiple interacting microservices. We discuss an initial prototype of JACKPOT along with a preliminary evaluation of this prototype based on experiments on a public container cloud.

## 1 Introduction

The ever-increasing need for companies to deliver frequent code changes to production for fixing problems, satisfying new requirements, and ultimately surviving in a software-driven market has fueled the adoption of agile software development practices, including *continuous deployment* [39] and *online experimentation*. Sophisticated algorithmic approaches [1, 2, 17, 21, 31] as well as mature systems [14, 36] are available today for devops engineers who wish to perform online experiments with Web and mobile applications.

Online experimentation of cloud microservices pose additional challenges. First, interactions between multiple cloud microservices that comprise an application can affect the overall user-perceived performance and correctness of the application [18–20, 42]. Hence cloud experimentation systems need to handle the practical complexities of routing traffic through specific combinations of microservice-versions (henceforth referred to as *paths*) during an experiment. Second, cloud environments are inherently volatile due to resource contention and infrastructure-related failures, both of which can have a profound impact on user experience [23], cause financial losses or damage reputation [4, 12, 13]. Cloud experimentation systems need to explicitly account for this by providing the engineer with an expressive framework for specifying what is the acceptable region of values for the key performance indicators (KPIs) of both business and system-wide interest. Third, the experimentation system should support different types of experiments based on the engineer's goals. The engineer might be interested in a) identifying the optimal combination of microservice-versions (paths) as quickly as possible, or in b) gradually shifting traffic to the optimal combination over the course of the experiment, or in c) passively observing the quality of different combinations of microservice-versions. While the distinction between these goals is well understood in the context of web and mobile experimentation [8, 16, 26, 28, 33, 37, 38], this is not the case for cloud experimentation. Current systems for cloud microservices experimentation [5, 6, 15, 32, 40, 43] developed prior to this work, fall short of meeting these challenges in a systematic manner. In particular, none of them discuss adaptive traffic shifting strategies in support of goals a) and b) when the cloud application consists of multiple interacting microservices.

Recent advances in cloud computing, including the maturity of container-based technologies (e.g., Docker [11]) and cloud infrastructure (e.g., Kubernetes [29]), as well as the advent of service meshes (e.g., Istio [24]), have created an opportunity to broaden the scope of online experimentation for cloud microservices. Service meshes enable 1) online experimentation for all microservices (even non-user-facing ones) and 2) a new breed of online experimentation that holistically considers multiple microservices and combinations of their versions (multivariate cloud experiments). The specific mechanism that enables 1) and 2) above is traffic management, through which the service mesh can be dynamically configured to control which microservice

versions are exposed and how the traffic is split across all versions. On the other hand, online experimentation on the cloud hinges on the ability to assess and compare versions of an individual microservice and version-combinations across microservices. To that end, service meshes collect a wealth of end-to-end KPIs (distributed tracing [35]) for individual microservices, including latency and inter-service call errors.

In this paper, we present JACKPOT, a system for online experiments for microservices which is specifically tailored to meet the unique challenges posed by cloud environments. JACKPOT comes with three key features. First, it is holistic in that it enables experimentation of multiple microservices; the winning variant it identifies in an experiment is a path comprising of multiple microservice-versions as opposed to an isolated version of an individual microservice. Second, the engineer can specify a reward KPI which is a target of maximization as well as constrained KPIs with limits imposed on them; the idea behind this feature is that the experiment may identify a path which maximizes reward within the subset of feasible paths that satisfy the constraints. These constraints are useful for capturing service level agreements (SLAs) on performance and correctness of the cloud application. Finally, JACKPOT provides a innovative way to combine the multiple KPIs into a single utility function; this function can be tuned so that constraints on KPIs can be interpreted as 'hard' (i.e., infeasible paths have zero utility) or 'soft' (i.e., violation of constraints leads to a penalty which depends on the extent of violation). Taken together, JACKPOT offers an unparalleled level of usability, expressivity, and flexibility to the devops engineer for online cloud experimentation.

## 2 Key Design Requirements

We now discuss the key requirements that arise during online experimentation of cloud microservices.

**Multivariate experiments:** A microservices application can exhibit complex, unpredictable emergent behavior due to interactions between its components [9,30,34]. As an example, Kaldor *et al.* [27] describe a scenario on Facebook.com, where the launch of a new feature in a frontend component led to the loss of predictive accuracy in a backend component that preemptively sends client-side resources, ultimately resulting in a 300ms (or 13%) increase in page load times. Other studies [18–20, 42] describe how interplay between services in large-scale applications could lead to not only sharp performance degradation, but also cascading failures and wide-spread application outages. Consequently, online cloud experiments need to identify the optimal combination of microservice versions (paths) as opposed to exclusively experimenting with individual services in isolation.

**Multi-KPI experiments:** The system-related metrics such as end-to-end latency can have a significant impact on business metrics. In particular, studies from Google [12], Akamai [4], Amazon [13] reveal that how an increase in system-related metric (latency) hurts customer conversion rates and revenue. The devops engineer must be able to express her preferences using multiple KPIs. This poses new requirements for modeling, algorithm design, and also the user interaction aspects. In the feature launch example, the devops engineer might prefer paths with tail latency within a given limit; among all such paths, she might prefer one which maximizes *click-through rate* (ctr), the proportion of users who click on a specific link in the front-end.

**Experiment goals:** Experiments can be classified into three types based on the engineer's goal: 1) *best-path identification*, where the engineer wishes to identify the optimal path with a pre-specified level of confidence, while minimizing the time required to do so, 2) *utility maximization*, where the engineer wishes to progressively shift traffic towards the optimal path during the course of an experiment, and 3) *pure estimation*, where the engineer wishes to discover statistically robust estimates of the quality of each path, which can be used to quantify regressions that might have been introduced by canary releases. We envision a system which supports all these goals. Note that experiments where the only variation is due to multiple versions of a single isolated microservice is a special case of our setup. Hence, they can also be handled by JACKPOT.

## 3 The JACKPOT System

We begin with a conceptual overview of JACKPOT which is illustrated in Figure 1. At the start of an experiment, the devops engineer declaratively describes an experiment using an *experiment spec* object, which includes a list of microservices and a list of path-level KPIs with associated constraints (Section 3.1). This user interaction is designed to be simple, interpretable and accessible to a wide spectrum of engineers who may not be experts in machine learning. JACKPOT transforms this experiment spec object to an internal representation, specifically a non-linear multivariate sigmoid (Section 3.2). This combines all the KPIs and constraints in the experiment spec into a scalar function which is learnt and optimized online during the experiment. Throughout the course of the experiment, JACKPOT maintains *belief* distributions that are associated with every (path, KPI) pair. The experiment is divided into periodic time epochs. By using epochs, we reduce the stress on the service mesh control plane due to reconfigurations. Further, we are able to utilize telemetry data from time series databases [22, 45] where data collection is not instantaneous but batched over windows of time.

At the start of each epoch, JACKPOT uses its current belief distributions to compute a probabilistic traffic policy using its Top-k Sigmoid Thompson Sampling algorithm (Section 3.4) and communicates the policy to the service mesh. At the end of each epoch, JACKPOT update its belief distributions (Section 3.3) based on values observed during this epoch for each (path-KPI) pair. In the rest of this section, we describe the above elements of JACKPOT in greater detail.

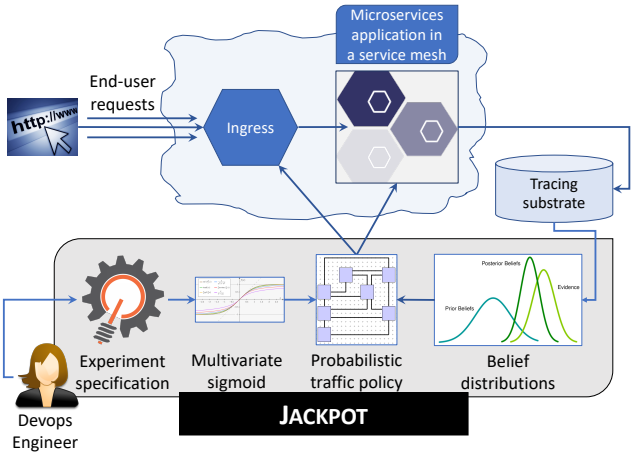



Figure 1: JACKPOT Optimized Service Mesh

## 3.1 Experiment Spec

A sample JACKPOT experiment spec in YAML format is illustrated on the right. There are three services in this experiment: all other services have a single fixed version during this experiment. The 'KPIs' section of the spec indicates that the engineer prefers paths with mean latency $\leq$500ms (constrained KPI) and among all such paths, she prefers the one which maximizes the mean ctr (reward KPI). Jackpot does not enforce routing requests to a specific service. Rather, it configures the Istio service mesh to split traffic across service versions. The rules are not exercised if no requests are sent to a particular service.

```
services:
  - reviews
  - ratings
  - details
KPIs:
  latency:
    limit: 500ms
  ctr:
    reward: True
```

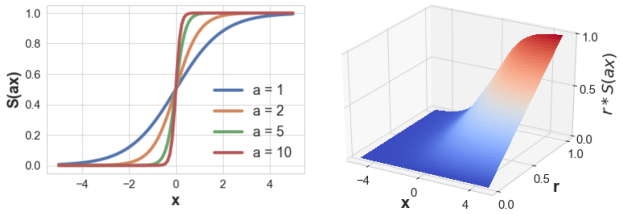
## 3.2 Multivariate Sigmoid Function

The experiment spec is internally represented in JACKPOT as a multivariate sigmoid function. In order to describe this, we need some basic mathematical notation which we introduce below. Let $X_0[p]$ denote the reward KPI for path $p$, let $X_1[p],\ldots,X_k[p]$ denote the constrained KPIs for path $p$, and let $\ell_1,\ldots,\ell_k$ denote their respective constraint limits. In the sample from Section 3.1, $X_0[p]$ denotes the ctr of path $p$, $k=1$, $X_1[p]$ denotes the latency of path $p$, and $\ell_1=500$. The *utility* of routing a request over path $p$ is defined as follows.

$$h_a(p) = \mathbb{E}[X_0[p]]\Pi_{j=1}^{k} S\left(a\left(1-\frac{\mathbb{E}[X_j[p]]}{\ell_j}\right)\right) \qquad (1)$$

In (1), $a$ denotes the *amplification factor*, a fixed positive constant which is a hyperparameter in JACKPOT and $S(y)$ is the logistic function, which belongs to the family of (S-shaped) sigmoid functions and defined as $S(ax)=\frac{1}{1+e^{-ax}}$.

The intuition behind this transformation is as follows. Suppose $a$ is sufficiently large (e.g., $a\geq 10$). Then $S(ax)$ acts like an indicator function: for positive and increasing values of $x$, $S(ax)$ rapidly approaches 1 and for negative and decreasing values of $x$, $S(ax)$ rapidly approaches 0. Hence, if the expected KPIs of a path $p$ are well within their respective limits, $h_a(p)$ equals the expected reward of $p$; otherwise, if even one of $p$'s expected KPIs significantly violates its limit, then $h_a(p)$ approaches 0. Suppose $a$ is relatively small (e.g., $a=1$). In this scenario, if $p$ violates a constraint, then it suffers a penalty which depends on the extent of the violation (i.e., $h_a(p)$ need not be close to 0 if the violation is not significant). In other words, the multivariate sigmoid function enables a high degree of flexibility in JACKPOT for modeling the engineer's preferences through suitable choice of the amplification factor. Observe that the above behavior of the multivariate sigmoid function is highly desirable since it directly corresponds to the preferences expressed by devops engineer in the experiment spec. Figures 2a and 2b illustrate the shapes of the logistic function (parameterized by $a$) and the multivariate sigmoid function (with $a=1$) respectively. In summary, this formulation achieves the following twin objectives: 1) it combines the multiple KPIs into a single utility function; 2) it enables constraints on KPIs to be interpreted as 'hard' or 'soft'.




(a) Logistic (b) Multivariate sigmoid

Figure 2: Sigmoid functions

## 3.3 Belief Updates

Equation (1) defines the utility of a path, but the expected KPIs $\mathbb{E}[X_j[p]]$ which are critical ingredients in (1) are unknown to JACKPOT in the beginning of an experiment and need to be estimated online in a statistically robust manner. JACKPOT accomplishes this by maintaining Bayesian belief distributions for $\mathbb{E}[X_j[p]]$ for all path-KPI pairs $p,j$, and updating them at the end of each epoch based on observations from all the past epochs. When the KPI pertains to a binary variable such as click-through, or conversion of an end-user, or occurrence of an error, JACKPOT uses the Beta-Bernoulli belief update model [10,41]. When the KPI pertains to a continuous variable such as latency or revenue, JACKPOT uses Beta updates when upper and lower bounds on the KPIs are known, and Gaussian updates when these bounds are unknown [3]. Since belief updates are standard machinery in Bayesian statistical inference, we refer the reader to [3, 10, 41] for further details.

The belief distributions serve two distinct purposes in JACKPOT. When the goal of the experiment is best-path identifi-

cation or utility maximization, the belief distributions allow JACKPOT to sample expected KPI values which act as surrogates for the true values. In all experiments (including those with pure estimation as the goal), the belief distributions allow engineers to derive statistically meaningful answers to questions such as 1) what is the probability of a specific path $p$ being the optimal path, and 2) what is the probability of a path $p$ satisfying all the constraints, and 3) what is the utility of path $p$.

### 3.4 Top-k Sigmoid Thompson Sampling

Continuous experimentation presents a fundamental tradeoff between exploration and exploitation which is best exemplified by the problem of multi-armed bandit. As an illustration, consider a gambler who enters a casino and faces a slot machine with multiple arms. When an arm is pulled, it produces a random payout drawn independently from some distribution. The objective of the gambler is to maximize the sum of payouts earned through a sequence of arm pulls. The basic dilemma confronting the gambler is whether to aggressively exploit an arm which is known to be the best one according to past observations (which may be few in number, and hence not statistically robust), or to aggressively explore the set of available arms (which implies exploring suboptimal arms and hence potential loss in utility). Extrapolating this idealized scenario to our case, the experimenter (gambler) wishes to maximize utility (payout) among various paths (arms). Thompson sampling is a heuristic for such explore/exploit problems, which chooses the arm probabilistically [44]. Our algorithm in this paper, the Top-k Sigmoid Thompson Sampling algorithm ($k$-STS), generalizes the classic Thompson sampling algorithm. We develop $k$-STS in order to account for multiple KPIs and various experimentation goals as discussed in Section 2.

JACKPOT uses $k$-STS to compute traffic splits in an adaptive manner at the beginning of each epoch based on observations so far. The intuition behind this algorithm is as follows. $k$-STS uses the belief distributions from Section 3.3 as surrogates for the true values of expected KPIs. Specifically, given a path-KPI pair $(p, j)$, $k$-STS samples a value from its corresponding belief distribution; it plugs in these sampled values into (1) to estimate the utility of path $p$. It then picks the top-k paths in terms of their estimated utility, and chooses one of them uniformly at random and declares this as the candidate path. $k$-STS repeats this selection procedure over multiple trials and computes the relative frequency with which path $p$ is declared as the candidate, which is then used as the traffic split for this epoch. For example, if path $p$ emerged as the top candidate during 35% of the trials, 0.35 will be the probability with which JACKPOT routes requests over path $p$ during this epoch.

When we set $k=1$, we obtain the 1-STS algorithm, which generalizes the classic Thompson sampling algorithm [44]. When we set $k=2$, we get the 2-STS algorithm, which generalizes the Top-2 Thompson sampling algorithm [37]. They are particularly useful in experiments with utility maximization and best-path identification as the respective goals.

## 4 Initial Prototype and Evaluation

We have implemented JACKPOT as a standalone Python service in our initial prototype. The JACKPOT service interacts with a cloud application consisting of Dockerized microservices running on a Kubernetes cluster. This interaction happens through the Istio service mesh [24] and the Jaeger end-to-end tracing framework [25]. At the beginning of each epoch of the experiment, JACKPOT transforms the traffic policy into Istio's virtual service configuration which is used by Istio's ingress and envoy proxies to enforce the policy. This configuration is tailored such that Istio Ingress injects a special HTTP request header called *jackpot-header* in each incoming end-user request. The path in which this request traverses is explicitly encoded as the value for this header. This request header, which is forwarded by all application services to upstream services, has a twin purpose: 1) it enables requests to be routed in accordance to the traffic split, and 2) it enables JACKPOT to collect path specific KPI observations for each request in an epoch using the Jaeger tracing substrate. JACKPOT currently uses the context provided by Jaeger to capture KPIs at a per-request level for competing paths.

In our experiments, we used the *bookinfo* microservices benchmark application [7] which is illustrated in Figure 3 along with a sample virtual service configuration. This application is an open-source microservice-based cloud application comprising four microservices, widely used by the Kubernetes and Istio community (22.1$k$ stars and 4$k$ forks on GitHub).

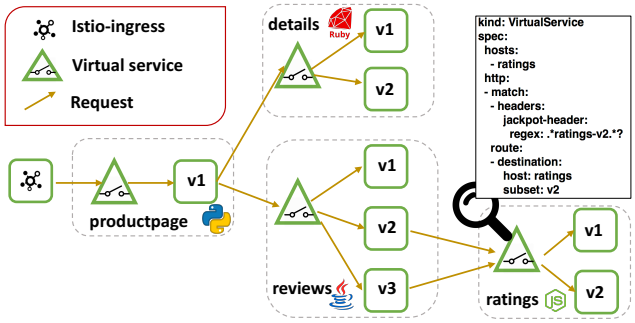



Figure 3: A snapshot of the bookinfo application extended with new microservice versions. The snapshot also illustrates inter-service REST API calls in this application. Also shown on top right is a sample virtual service configuration.

**Evaluation:** We experimentally evaluate the performance of 1-STS, 2-STS , and (UNIF), which splits traffic equally across all paths. UNIF is a natural point of comparison since it is the most commonly used strategy in practice.

We consider a setting with 10 paths as depicted in Table 1. We use Istio's fault injection capabilities to create latency variations in microservice versions [24]. We specify a

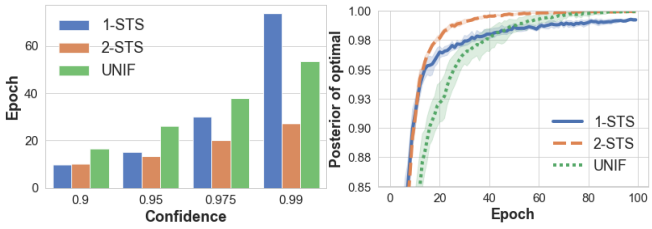
constraint in the experiment which requires mean latency to be $\leq \ell_1 = 300ms$ (i.e., $\mathbb{E}[X_1[p]] \leq 300ms$). This results in five feasible paths in the application. Further, we introduce synthetic rewards for each path (e.g., click-through rate). The mean reward for each path is sampled from the uniform distribution (U[0,1]) at the start of each experiment. We set the amplification factor $a = 10$ for the experiments so that latency constraint is interpreted as a hard constraint by JACKPOT (Section 3.2). Each run of the experiment consists of 100 epochs, and we send end-user requests direct at the product-page service at a mean rate of 50-per-epoch. We conducted five runs and report on results averaged from these runs.

Table 1: Path characteristics. Services *pp*, *det*, *rev* and *rat* correspond to productpage, details, reviews and ratings in Figure 3. The optimal path is bolded and colored in red.

| *path* | $\mathbb{E}[X_0[p]]$ | $\mathbb{E}[X_1[p]]_{ms}$ | $h_a(p)$ |
|---|---|---|---|
| $pp_{v1}$,$det_{v2}$,$rev_{v2}$,$rat_{v1}$ | 0.21 | 60 | 0.20 |
| $pp_{v1}$,$det_{v1}$,$rev_{v3}$,$rat_{v1}$ | 0.88 | 470 | 0.003 |
| $pp_{v1}$,$det_{v2}$,$rev_{v2}$,$rat_{v2}$ | 0.22 | 260 | 0.17 |
| $pp_{v1}$,$det_{v1}$,$rev_{v1}$ | 0.92 | 456 | 0.005 |
| $pp_{v1}$,$det_{v1}$,$rev_{v3}$,$rat_{v2}$ | 0.48 | 666 | 0 |
| $pp_{v1}$,$det_{v1}$,$rev_{v2}$,$rat_{v2}$ | 0.6 | 450 | 0.004 |
| **$pp_{v1}$,$det_{v2}$,$rev_{v1}$** | **0.77** | **253** | **0.64** |
| $pp_{v1}$,$det_{v2}$,$rev_{v3}$,$rat_{v2}$ | 0.52 | 460 | 0.002 |
| $pp_{v1}$,$det_{v2}$,$rev_{v3}$,$rat_{v1}$ | 0.28 | 260 | 0.22 |
| $pp_{v1}$,$det_{v1}$,$rev_{v2}$,$rat_{v1}$ | 0.20 | 260 | 0.16 |

**Best-path identification:** Let $\Pr[p = p^*]$ denote the posterior probability of a fixed path $p$ being the optimal path – this probability (henceforth referred to as posterior of $p$) can be estimated from the belief distributions as described in Section 3.4. Figure 4a displays the average number of epochs required by the different algorithms in order for the posterior of $p^*$, the true optimal path, to reach various levels of confidence. 1-STS seems to reach the 0.9 confidence level as rapidly as 2-STS and both of them outperform UNIF. However, 1-STS struggles to reach higher levels of confidence since it shifts much of its traffic towards the optimal version at the expense of refining its estimates about the suboptimal versions. Conversely, 2-STS does not exclusively focus on a single candidate path. Thus, in order to reach the .99 confidence level, 2-STS requires 49% fewer epochs compared to UNIF, and 63% fewer epochs compared to 1-STS in our experiment. Figure 4b provides a more comprehensive view of the observation made in Figure 4a. In this figure, we report the posterior of $p^*$ at each epoch of the experiment. 2-STS outperforms both 1-STS and UNIF in terms of the number of epochs needed to reach reaching a given level of confidence.

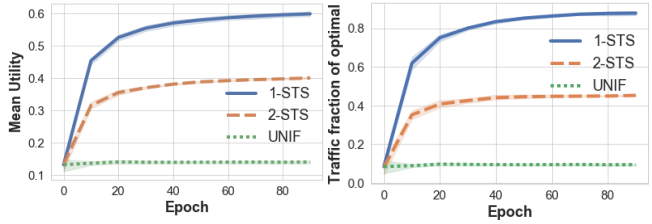
**Utility maximization:** We report the mean utility and the ratio of requests routed through the optimal path by each of our algorithms. The optimal path $(pp_{v1}, det_{v2}, rev_{v1})$ has a mean utility of 0.64 in our set up (Table 1). From figure 5a,




(a) # of epochs per level (b) Posterior per epoch

Figure 4: Best path identification experiment

we observe that 1-STS maximizes the mean utility during experimentation and converges towards the optimal value. Figure 5b depicts the ratio of requests routed through the optimal path. 1-STS gradually shifts traffic towards the optimal and outperforms others in terms of reward.




(a) Mean Utility (b) Traffic split

Figure 5: Utility maximization

We conclude this section by recalling that both 1-STS and 2-STS are special cases of $k$-STS. The fact that our $k$-STS meta-algorithm can be effectively applied to both best-path identification and utility maximization merely be setting the hyperparameter $k$ appropriately is one of the clean features of JACKPOT.

## 5 Research Challenges

The main motivation for cloud-native applications to follow the microservice architectural style, rather than traditional monoliths, is to increase overall agility by enabling teams to develop and operate different (loosely coupled) microservices, with their own release schedules. Our goal is to empower these individual teams to deliver code even more frequently to experiment aggressively. Thus, it is of paramount importance to dynamically incorporate versions of microservices as they arrive into ongoing experiments.

Cloud applications and environments can vary in terms of their path-level traffic splitting and telemetry functionality. We use distributed tracing in our prototype, which explicitly records per-path KPIs; this functionality may not always be enabled within an application. The ability to handle heterogeneous cloud environments is a major challenge from an algorithm design and systems perspective.

## 6 Discussion for HotCloud 2020

Underlying this preliminary work is an ambitious goal. We hope to unleash the power of online experimentation to an unprecedented level through the novel, statistically robust algorithms suitable for developers of cloud-native microservice-based applications. We are aware of the fact that many assumptions and aspects of our work can be perceived as controversial. We list some of them below and enthusiastically invite reviewer feedback. They also present excellent opportunities for lively discussions at HotCloud.

**Online experimentation formulation.** Perhaps, the most controversial aspect of the multivariate cloud experimentation approach in JACKPOT is the consolidation of multiple KPIs of interest into a single multivariate sigmoid function. There are alternative approaches, with advantages and disadvantages, that could be considered. We expect this topic to be a source of discussion.

**Scalability.** Multivariate experimentation is inherently limited by the fact that, as the number of variants (competing paths) increases, the sample complexity, or equivalently, the amount of time required by the experiment to reach statistically robust conclusions can also increase significantly. This would not prevent JACKPOT from working; however, it would result in longer experiments. If there is a need for shorter experiments, one of the suggested practices for experimenters would be to limit competing variants to a subset of paths. This can be done by performing API based experimentation. An individual API generally comprises fewer microservices as opposed to the whole system, and thus gives rise to fewer possible competing variants. Devops engineers can perform various API experiments in parallel using JACKPOT.

**Absence of datasets.** Unlike supervised and machine learning problems, open and publicly available datasets for researchers interested in continuous experimentation systems do not exist. A broader community discussion on this topic could be the catalyst for joint efforts to make such data available and to create proper benchmarks for service meshes and microservice-based cloud applications.

## References


[1] Deepak Agarwal. Computational advertising: The linkedin way. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, CIKM '13, page 1585–1586, New York, NY, USA, 2013. Association for Computing Machinery.

[2] Deepak Agarwal, Bo Long, Jonathan Traupman, Doris Xin, and Liang Zhang. Laser: A scalable response prediction platform for online advertising. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, page 173–182, New York, NY, USA, 2014. Association for Computing Machinery.

[3] Shipra Agrawal and Navin Goyal. Near-optimal regret bounds for thompson sampling. *J. ACM*, 64(5), September 2017.

[4] Akamai. Akamai online retail performance report: Milliseconds are critical. https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp, Apr 2017.

[5] Dan Ardelean, Amer Diwan, and Chandra Erdman. Performance analysis of cloud applications. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 405–417, Renton, WA, April 2018. USENIX Association.

[6] Argo Rollouts. https://argoproj.github.io/argo-rollouts/, 2019.

[7] Bookinfo. https://istio.io/docs/examples/bookinfo/, 2020.

[8] Hyeong Soo Chang. An asymptotically optimal strategy for constrained multi-armed bandit problems. *Mathematical Methods of Operations Research*, Jan 2020.

[9] Carlos E. Cuesta, Elena Navarro, and Uwe Zdun. Synergies of system-of-systems and microservices architectures. In *Proceedings of the International Colloquium on Software-Intensive Systems-of-Systems at 10th European Conference on Software Architecture*, SiSoS@ECSA '16, New York, NY, USA, 2016. Association for Computing Machinery.

[10] Morris H. DeGroot and Mark J. Schervish. *Probability and Statistics*. Addison-Wesley, 3 edition, 2002.

[11] Docker. https://www.docker.com/, 2020.

[12] Yoav Evinav. Marissa mayer at web 2.0. http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html/, Nov 2006.

[13] Yoav Evinav. Amazon found every 100ms of latency cost them 1 https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/, Jan 2019.

[14] Firebase. https://firebase.google.com, 2020.

[15] Flagger. https://docs.flagger.app, 2019.

[16] Aditya Gopalan, Shie Mannor, and Yishay Mansour. Thompson sampling for complex online problems. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 100–108, Bejing, China, 22–24 Jun 2014. PMLR.

[17] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 13–20, Madison, WI, USA, 2010. Omnipress.

[18] Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, and et al. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, page 1–14, New York, NY, USA, 2014. Association for Computing Machinery.

[19] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. Why does the cloud stop computing? lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, page 1–16, New York, NY, USA, 2016. Association for Computing Machinery.

[20] Zhenyu Guo, Sean McDirmid, Mao Yang, Li Zhuang, Pu Zhang, Yingwei Luo, Tom Bergan, Madan Musuvathi, Zheng Zhang, and Lidong Zhou. Failure recovery: When the cure is worse than the disease. In *Presented as part of the 14th Workshop on Hot Topics in Operating Systems*, Santa Ana Pueblo, NM, 2013. USENIX.

[21] Daniel N. Hill, Houssam Nassif, Yi Liu, Anand Iyer, and S.V.N. Vishwanathan. An efficient bandit algorithm for realtime multivariate optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1813–1821, New York, NY, USA, 2017. Association for Computing Machinery.

[22] InfluxDB. https://www.influxdata.com/products/influxdb-overview/, 2013.

[23] Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. On the performance variability of production cloud services. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 104–113, Washington, DC, USA, 2011. IEEE Computer Society.

[24] Istio. https://istio.io/docs/concepts/what-is-istio/, 2020.

[25] Jaeger. https://www.jaegertracing.io, 2020.

[26] Ramesh Johari, Pete Koomen, Leonid Pekelis, and David Walsh. Peeking at a/b tests: Why it matters, and what to do about it. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1517–1525, New York, NY, USA, 2017. Association for Computing Machinery.

[27] Jonathan Kaldor, Jonathan Mace, Michał Bejda, Edison Gao, Wiktor Kuropatwa, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, and et al. Canopy: An end-to-end performance tracing and analysis system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 34–50, New York, NY, USA, 2017. Association for Computing Machinery.

[28] Ron Kohavi and Roger Longbotham. *Online Controlled Experiments and A/B Testing*, pages 922–929. 01 2017.

[29] Kubernetes. https://kubernetes.io/, 2020.

[30] Tobias Kunze. Complex emergent behaviors in organic microservice architectures. https://glasnostic.com/blog/complex-emergent-behaviors-organic-microservice-architectures, Jul 2019.

[31] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 661–670, New York, NY, USA, 2010. Association for Computing Machinery.

[32] Hongqiang Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. CrystalNet: Faithfully emulating large production networks. In *SOSP '17 Proceedings of the 26th Symposium on Operating Systems Principles*, pages 599–613. ACM, October 2017.

[33] Benedict C. May, Nathan Korda, Anthony Lee, and David S. Leslie. Optimistic bayesian sampling in contextual-bandit problems. *J. Mach. Learn. Res.*, 13:2069–2106, June 2012.

[34] Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015.

[35] Opentracing. https://opentracing.io/, 2020.

[36] Optimizely. https://www.optimizely.com, 2020.

[37] Daniel Russo. Simple bayesian algorithms for best arm identification. In Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1417–1418, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR.

[38] Daniel Russo and Benjamin Van Roy. An information-theoretic analysis of thompson sampling. *J. Mach. Learn. Res.*, 17(1):2442–2471, January 2016.

[39] Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. Continuous Deployment at Facebook and OANDA. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 21–30, New York, NY, USA, 2016. ACM.

[40] Gerald Schermann, Dominik Schöni, Philipp Leitner, and Harald C. Gall. Bifrost: Supporting continuous deployment with automated enactment of multi-phase live testing strategies. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, pages 12:1–12:14, New York, NY, USA, 2016. ACM.

[41] Mark J. Schervish. *Theory of Statistics*. Springer-Verlag, 1995.

[42] Riza Oktavian Nugraha Suminto. *Mitigating Cascading Performance Failures and Outages in Cloud Systems*. PhD thesis, The University of Chicago, https://knowledge.uchicago.edu/record/2095?ln=en, 12 2019.

[43] Alexander Tarvo, Peter F. Sweeney, Nick Mitchell, V.T. Rajan, Matthew Arnold, and Ioana Baldini. Canaryadvisor: A statistical-based tool for canary testing (demo). In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, page 418–422, New York, NY, USA, 2015. Association for Computing Machinery.

[44] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 12 1933.

[45] What is Prometheus? https://prometheus.io/docs/introduction/overview//, 2012.