**SOCC’15 Final Paper Revision Plan**

**Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski (UCSB)**

**Shephard: Olga Papaemmanouil (Brandeis Univ.)**

Below we overview our modification plan for our SOCC paper submission. In bold, we summarize a reviewer comment, question or suggestion. Following each bold review sentence, we outline the steps that we plan to take to address the reviewer’s feedback. We thank the reviewers and our shepherd for their help in improving this paper and preparing it for publication.

If there are any additional changes required or suggested, please let us know. We are happy to discuss these or any of the below with you further. Thanks again.

**Empirical findings in section 2 are very interesting. Briefly mention them in the introduction. [DONE]**

We will add a sentence to the introduction emphasizing the significance of our findings in section 2.

Section 1.0: Cerebro uses static analysis to identify the cloud SDK invocations that dominate the response time of web APIs. By surveying a collection of web applications developed for a PaaS cloud, we show that such applications indeed spend majority of their execution time on cloud SDK invocations. Further, they do not have many branches and loops, which makes them amenable to static analysis.

**Load conditions used for benchmarking (1 req/min) is too small -- How would the results change, if the actual load was 30 req/min? [DONE]**

We assume that the time series of cloud SDK benchmarking data is ergodic (i.e. stationary over a long period). Under that assumption, QBETS is insensitive to the measurement frequency (at the cloud SDK monitor) so the correctness and tightness should not be affected by a higher measurement frequency. However, an increase in the frequency by a factor of 30 might foreshorten the duration over which the SLAs are durable. Examining the data shows that the predicted percentile values change most dramatically as a result of change points, and a higher frequency would not change the time between change points (since the changes are likely exogenous). Thus we believe that the results are valid for a higher frequency measurement series.

As for measuring the actual response time of the APIs under a higher load, we assume that the cloud platform is scalable thereby distributing the API load across many physical nodes. This is true for most real world cloud platforms, including Google App Engine and AppScale. The distribution of API load (sometimes coupled with Autoscaling features of the cloud platform), helps maintain API response time steady regardless of the load conditions.

We will address the comment by adding these analytical arguments to the paper.

Section 4.1: Our test request rate (1 request/minute) is not sufficient to put the backend cloud servers under any stress. However, cloud platforms like Google App Engine and AppScale are highly scalable. When the load increases, they automatically spin up new backend servers, and maintain the average response time of deployed web APIs steady. This enables us to measure and evaluate the correctness of the Cerebro predictions under light load conditions. Note that our cloud SDK benchmarking rate at the cloud SDK monitor is also 1 request per minute. We assume that the time series of cloud SDK performance is ergodic (i.e. stationary over a long period). Under that assumption, QBETS is insensitive to the measurement frequency, and a higher benchmarking rate would not significantly change the results.

**Can we show that the system can recognize highly unpredictable code (maybe a loop based on a randomly generated value?) and loosen its performance estimates appropriately? [DONE]**

We do not know the relationship between code predictability and the prediction results -- only that the codes we examined for GAE do not exhibit this level of random execution behavior. We will address this comment by pointing out a possible investigation of this relationship as part of our future work.

Could not address, since the future work section was removed due to space limitations.

**The main contribution of the paper (QBETS) has been published before [DONE]**

It is true that QBETS has been published before but it has not been applied in this context. We include in the paper a discussion of both the origin of QBETS and the contribution that we make with this paper to cloud computing.

Sections 3.3 and 3.4 outlines our unique approach for using QBETS.

**For private cloud offerings, where operators may have no limitation of how long a service can run, we may see different execution paths appearing in the applications. [DONE]**

Cerebro takes an exhaustive approach when making SLA predictions, where it considers "all" the branches in the code. It makes separate predictions for each possible branch, and picks the one with the highest value as the final SLA. Therefore, Cerebro already copes with applications with many execution paths. The Stocks application used in our empirical evaluations puts this feature to the test. We will make this clear in our writing.

Section 3.4: Cerebro uses the largest predicted value (across path sequences)

as its SLA prediction for a web API operation. The exhaustive approach by

which Cerebro predicts SLAs for all possible program paths ensures that

the final SLA holds valid regardless of which path gets executed at runtime.

**Currently Cerebro benchmarks SDK operations on some synthetic datasets. It may make sense to modify the applications so that themselves are able to collect information on how the operations are running on their own data. [DONE]**

We believe that the use of application instrumentation would likely improve the quality of the results or, at worst, leave them unchanged. One goal of the work is to determine the degree to which a non-invasive approach (i.e. one that does not require modification of the application for instrumentation purposes) can be made successful. To address this comment, we will add an investigation of the effects (likely positive) of application instrumentation to future work.

Could not address, since the future work section was removed due to space limitations.

**What happens if there is a lot of variance in the behavior of the system (no stable behavior at all)? [DONE]**

QBETS requires a minimum number of consecutive measurements (determined from the confidence bound specified and the percentile of interest) to make a prediction. Call this the "measurement set." The variance of the measurement set can be high and QBETS predictions will be correct. However the series must be approximately stationary (i.e. ergodic) meaning that the mean and the variance cannot change abruptly. If they do, QBETS attempts to detect the change and adapt, but the speed with which is does so is bounded by the minimum measurement set required. Thus if changes due to non-stationarity occur more frequently than a complete measurement set can be gathered, QBETS prediction quality will suffer. This degradation will be reflected in the correctness and tightness measures of prediction quality. To address this issue we will attempt to add a clarification to the paper but also to point the interested reader to the relevant exposition of QBETS in the literature.

Section 3.3: The minimum history length also provides a bound on the variability of the time series that can be tolerated by QBETS. In general, each time series must be

approximately ergodic, meaning their mean and the variance should not change abruptly. More specifically, if the values in the time series change too fast for QBETS to gather a stationary dataset at least as long as the minimum history length, its prediction accuracy may suffer.

**How to decide the duration of history data? [DONE]**

Given the percentile (p) and confidence (c), a theoretical lower bound for the history length can be calculated easily. For c = 0.01 and p = 0.95 this lower bound is around 90 (i.e. 90 data points in the input time series). There is no upper bound for history length, but it makes sense to keep a history of several thousand data points (i.e. a few days of monitoring data). We will briefly mention this in the paper.

Section 3.3: For example, predicting the $95^{th}$ percentile of the API execution time, with an upper confidence of $0.01$ requires at least $90$ observations. We use this limit as a lower bound for the length of the history to keep. There is no upper bound for the history length that QBETS can process. But in Cerebro's case, several thousand data points in the history (i.e. 1-3 days of monitoring data) provides a good balance between results accuracy and computation overhead.

**How are the 35 applications chosen? [DONE]**

We crawled GitHub for Google App Engine apps, and picked a set that was well documented, builds and runs without errors. We will discuss these steps in the methodology section.

Section 2.0: These web APIs are open source (available via GitHub~\cite{github}), written in Java, and run over Google App Engine or AppScale without modification.

We selected them based on availability of documentation, and the ability to compile and run them without errors.

**Figures 2 and 3 have a lot of extra space. The y-axis for each can start from 0.6 and 0.4 respectively. [DONE]**

Will make the changes appropriately.

Figures updated accordingly.

**In section 3.2 how are the data sizes decided? [DONE]**

These were determined based on the type of experiments we were planning to carry out using the prototype. However, out system is easily extensible to add benchmarks with other data sizes if necessary (e.g. 10,000 or 1000,000 entities). We will mention this fact in the paper.

Section 3.2: In addition, it benchmarks loop iteration over datastore entities to capture the performance of iterative datastore reads for datastore result set sizes of 10, 100, and 1000. We limit ourselves to these values because the PaaS requires that all operations complete (respond) within 60 seconds -- so the data sizes returned are typically small. Sizing up the datastore in terms of powers of 10, mirrors the typical approach taken by DevOps personnel to approximate the size of a database. If necessary, our prototype allows adding iterative datastore read benchmarks for other result set sizes easily.