Service-Level Agreement Durability for Web Service Response Time

Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski
Department of Computer Science
Univ. of California, Santa Barbara

***Abstract*—Cloud computing is an attractive model for deploying web services in a highly scalable manner. Users access such cloud-hosted services via their web-facing application programming interfaces (APIs). Prior work has shown that it is possible to use a combined approach of static analysis and cloud platform monitoring to predict the response time upper bounds of such web APIs. This technique can be employed to automatically generate service level agreements (SLAs) concerning the performance of cloud-hosted web APIs.**

In this work, we explore the validity period of auto-generated SLAs in cloud settings. We discuss a simple model by which API consumers can establish a response time SLA with the cloud platform, and renegotiate it when/if the SLA becomes invalid due to the dynamic nature of the cloud. Using empirical methods and simulations on a real world public cloud platform, we show that it is possible to auto-generate highly durable response time SLAs for cloud-hosted web APIs, thereby keeping the number of SLA invalidations and renegotiations very low, over long periods.

***Keywords*-Web APIs; Performance; Service Level Agreements; Cloud Computing;**

I. Introduction

Today, web services are an essential technology for implementing complex distributed applications. They promote modularity and software reuse, while leveraging the scalability features and maintenance provided by others. For these reasons developers increasingly integrate remote, Internet-accessible web services via web application programming interfaces (web APIs) into their web, cloud, and mobile applications. The benefits to programmer productivity that such development practices permit have resulted in a vast number and diversity of available web APIs [1].

Unfortunately, reusing existing services has its drawbacks. In particular, web APIs impact the performance, behavior, and reliability of the applications that integrate them. Web service functionality and performance can change over time without prior notice, while their APIs remain unchanged. Moreover, there is a shortage of tools to help developers reason about the impact of web API dependencies, and their potential for dynamic change throughout an application's lifecycle (i.e. development, deployment, and runtime).

In this paper, we study the use of on-line monitoring and statistical forecasting of web API performance as a means of providing each API consumer with a guaranteed minimum performance level. More specifically, we explore the idea of formulating Service Level Agreements (SLAs) for web APIs, based on automatic prediction of web API response times. Most cloud platforms which are used to serve web APIs today only offer probabilistic SLAs for service availability but not response time. Our work focuses on response time SLAs for web APIs deployed in Platform-as-a-Service (PaaS) clouds.

We use the term "SLA" to refer to the minimum service level promised by the service provider regarding some non-functional property of the service such as its availability or performance (response time). Such SLAs are explicitly stated by the service provider, and are associated with a correctness probability, which can be described as the likelihood the service will meet the promised minimum service level. An availability SLA that follows this notion takes the form: "the service will be available $p\%$ of the time". Here the value $p\%$ is the correctness probability of the SLA. Similarly, a response time SLA would take the form of the statement: "the service will respond under Q milliseconds, $p\%$ of the time. Naturally, p should be a value close to 100, for this type of SLAs to be useful in practice.

In a corporate setting, an SLA would consist of additional clauses describing what happens if and when the service fails to meet the promised minimum service level (for example, if the service is only available $p'\%$ of the time, where $p' < p$). This typically boils down to service provider paying some penalty (a refund), or providing some form of free service credits for the users. We do not consider such legal and social obligations of an SLA in this work, and simply focus on the minimum service levels and the associated correctness probabilities, since those are the things that matter from a performance and capacity planning point of view of an application. Some authors use the term Service Level Objective (SLO) to refer to promised minimum service levels, and reserve the term SLA for the aggregate of SLOs and other action clauses [2]. Since we are only looking at the minimum service levels, we disregard this separation, and make liberal use of the term SLA.

To enable our study, we have developed Cerebro [3], a system that predicts the response time of web APIs hosted in a PaaS cloud. Cerebro is able to determine automatically the bounds on API response time with a specific correctness probability. Our previous work [3] describes the effectiveness of Cerebro when it is used in conjunction with Google's public PaaS (Google App Engine) and the AppScale [4] private on-premise PaaS.

In this work, we explore the use of Cerebro predictions as the basis of SLAs between an API consumer, or client, and a service hosted by the PaaS. Specifically, we detail the duration over which SLAs offered to clients of applications running in Google App Engine persist. The SLAs based on predicted API response times do not remain valid indefinitely due to the dynamic performance variations that occur in the underlying cloud platform. Empirical analysis on cloud workload traces has shown that production cloud platforms often display performance variations, sometimes with temporal patterns [5].

The resulting SLA duration is an important parameter because PaaS API consumers often wish to contract for specific minimum service level guarantees, and must renegotiate when those guarantees expire or can no longer be sustained. This work demonstrates that, using a combination of online benchmarking and static program analysis, Cerebro can generate SLAs that are *durable* over long periods. That is, using Cerebro it is possible for a user of Google App Engine or AppScale to offer statistically reliable response time SLAs on web APIs that persist over long periods (and thus do not require frequent renegotiation).

To analyze SLA durability and how API consumers are impacted by it, we perform extensive testing and empirical evaluation of Google App Engine using a set of open source Java web applications. We also employ simulation to explore different options for SLA renegotiation. Our results indicate that on average, the minimum duration for which Cerebro SLAs remain valid for this PaaS is 12 days. We also show that over a period of 112 days, the maximum number of times that any API consumer must renegotiate their SLA is 6. Furthermore, we find that in some cases Cerebro prompts an API consumer to renegotiate an SLA when the predicted new SLA value is very close to the invalidated SLA value. Such renegotiations are not useful in practice, and only serve to increase the renegotiation overhead for API consumers. We thus also present a threshold-based mechanism that both reduces the number of required renegotiations, and extends SLA validity duration. To our knowledge, no other research or system is able to predict durable performance SLAs for applications hosted on a public PaaS such as Google App Engine.

II. Cerebro

In this section we provide an overview of PaaS clouds and Cerebro. Then we present a model for negotiating response time SLAs for web APIs deployed in PaaS clouds.

A. *Properties of PaaS-hosted Applications*

PaaS clouds enforce a restricted programming model on the application developer to guarantee the scalability, security and the availability of the cloud-hosted applications. PaaS clouds provide a predefined set of programming interfaces through which they export various platform services. We shall refer to these programming interfaces as the cloud software development kit (cloud SDK). The cloud SDK exposes scalable functionality that can be used to program a wide range of application features. These include key-value data stores, databases, caching, task scheduling, and user management. In a typical PaaS environment such as Google App Engine, AppScale [4], and Microsoft Azure, developers must use the cloud SDK to implement the required application functionality.

Similarly, PaaS clouds may impose restrictions on performing certain types of I/O operations, and executing long running tasks [6], [7], [8]. For example, Google App Engine denies applications access to the local file system (i.e. no file I/O). Furthermore, it forces the developer to implement all application tasks as request-response interactions of a web service where all requests must be processed under 60 seconds. Any task that takes longer than this is terminated by the cloud platform. This restriction is particularly interesting to us, since it gives a 60 second default SLA for all web APIs developed for Google App Engine. The result of all these restrictions is a programming model that is amenable to static analysis, a feature we exploit in the design of Cerebro. By surveying a collection of open source PaaS applications we have also found that program features that typically inhibit static analysis (e.g. excessive branching and loops) are rare among PaaS-hosted applications.

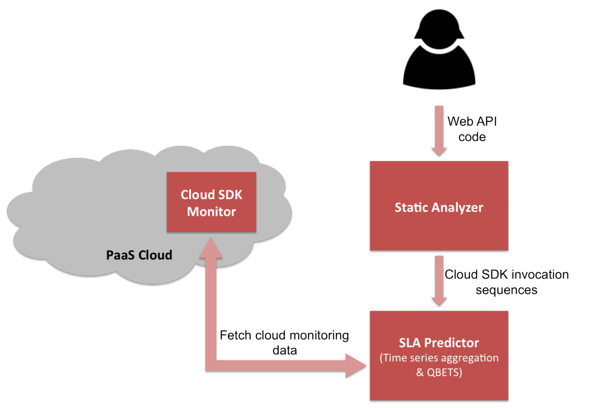
Web API code
Cloud SDK Monitor
PaaS Cloud
Static Analyzer
Cloud SDK invocation sequences
Fetch cloud monitoring data
SLA Predictor
(Time series aggregation & QBETS)

Figure 1. Main components of Cerebro and their interactions.

B. *Cerebro Architecture and Statistical Model*

Figure 1 illustrates the main components of Cerebro, and how they interact with each other. Cerebro runs a cloud SDK monitor in the PaaS cloud, that periodically benchmarks and records the execution time of each cloud SDK (PaaS service) operation. This component executes continuously, and separately from all other cloud-hosted applications.

When an application developer *deploys* a new application to the cloud platform (i.e. before the application begins executing), Cerebro automatically intercepts this process, and statically analyzes the application. The analyzer extracts the sequence of cloud SDK operations invoked by each web

API operation in a given application. When the application code contains branches, it extracts multiple sequences of cloud SDK operations – one sequence per code path. The static analyzer also looks for loops, and if any cloud SDK invocations are embedded within a loop, it attempts to estimate loop bounds using existing loop bound analysis methods [9]. All this information is then passed to the SLA predictor.

Cerebro's SLA predictor contacts the cloud SDK monitor to retrieve the gathered benchmarking data pertaining to the cloud SDK operations used in the application. The predictor aggregates this data for the longest sequence of cloud SDK calls (path through the operation), and forms a single time series from benchmark results. Cerebro then processes this aggregate time series using QBETS (Queue Bounds Estimation from Time Series) [10], a non-parametric time series analysis and forecasting technique. QBETS analyzes the given time series, and predicts an upper bound for its p-th percentile, where p is configurable. The predicted value Q can be used to form a response time SLA of the form "the web API operation responds under Q milliseconds at least p percent of the time".

Cerebro only takes the performance of the cloud SDK calls into account when making predictions. It ignores all other operations in the web API code, and assumes that the response time of a PaaS-hosted web API is primarily determined by the cloud SDK calls. Our previous work has shown that this is a reasonable assumption – i.e. PaaS-hosted web APIs spend most of their time ($> 90\%$) executing cloud SDK calls. Also the conservative nature of QBETS helps offset any discrepancies that might occur due to operations other than cloud SDK calls.

For a given web API, Cerebro predicts an initial response time SLA at the API's deployment-time. It then consults an on-line API benchmarking service to continuously verify the predicted response time SLA to determine if and when it has been violated. SLA violations occur when conditions in the PaaS change in ways that adversely impact the performance of the cloud SDK operations. Such changes can result from congestion (multitenancy), component failures, and modifications to PaaS service implementations. The continuous tracking of SLA violations is necessary to notify the affected API consumers promptly.

Cerebro also periodically recomputes the SLAs for the APIs over time. Cerebro is able to perform fast, online prediction of time series percentiles via QBETS as more SDK benchmarking data becomes available from the cloud SDK monitor. This periodic recomputation of SLAs is important because changes in the PaaS can occur that make new SLAs available that are better and tighter than the previously predicted ones. Cerebro must detect when such changes occur so that API consumers can be notified and SLAs renegotiated.

To determine SLA durability, we extend Cerebro with a statistical model for detecting when a Cerebro-generated SLA becomes invalid. Suppose at time t Cerebro predicts value Q as the p-th percentile of some API's execution time. If Q is a correct prediction, the probability of API's next measured response time being greater than Q is $1-(0.01p)$. If the time series consists of independent measurements, then the probability of seeing n consecutive values greater than Q (due to random chance) is $(1-0.01p)^n$. For example, using the 95^{th} percentile, the probability of seeing 3 values in a row larger than the predicted percentile due to random chance is $(0.05)^3 = 0.00012$.

This calculation is conservative with respect to autocorrelation. That is, if the time series is stationary but autocorrelated, then the number of consecutive values above the 95^{th} percentile that correspond to a probability of 0.00012 is larger than 3. For example, in previous work [10] using an artificially generated AR(1) series, we observed that 5 consecutive values above the 95^{th} percentile occurred with probability 0.00012 when the first autocorrelation was 0.5, and 14 when the first autocorrelation was 0.85. QBETS uses a look-up table of these values to determine the number of consecutive measurements above Q that constitute a "rare event" indicating a possible change in conditions.

Each time Cerebro makes a new prediction, it computes the current autocorrelation and uses the QBETS rare-event look-up table to determine C_w: the number of consecutive values that constitute a rare event. We measure the time from when Cerebro makes the prediction until we observe C_w consecutive values above that prediction as being the time duration over which the prediction is valid. We refer to this duration as the *SLA validity duration*.

C. SLA Negotiation Model

We extend Cerebro with an SLA negotiation process that invalidates SLAs at the end of the SLA validity duration, and re-establishes a new SLA for the API consumer. API consumers acquire an initial SLA for a web API hosted by a Cerebro-equipped PaaS as part of the API subscription process (i.e. when obtaining API keys). At this point Cerebro also records the tuple $< API\ Consumer, API, Timestamp, SLA\ Value >$.

When Cerebro detects consecutive violations of one of its predictions, it considers the corresponding SLA to be invalid, and notifies the affected API consumers to establish a new SLA. Cerebro can suspend API access by affected API consumers until renegotiation occurs, or simply record violations for later remediation. Ideally however, it is desirable that the SLA be immediately and automatically renewed. We refer to such on-the-fly SLA changes as SLA renegotiations. Upon renegotiation, Cerebro updates the *Timestamp* and *SLA Value* entries in the appropriate data tuple for future reference.

There is also a second type of SLA renegotiation that is possible with Cerebro. When recomputing SLAs periodically, Cerebro might come across situations where the latest SLA is smaller than some previously established SLA (i.e. a tighter SLA is available). Cerebro can notify the API consumer about this prospect, but wait for authorization from the API consumer before establishing the SLA. If the API consumer consents to the SLA change, Cerebro may update the data tuple, and treat the SLA as established. We next use empirical testing and simulations to explore the feasibility of the Cerebro SLA negotiation process, and evaluate how response time SLA duration and invalidation impact API consumers over time.

III. Methodology

In this section we describe the experimental methodology we use to evaluate how the SLAs generated by Cerebro change over time. Our goal is to understand the frequency with which Cerebro requires API consumers to renegotiate auto-generated performance SLAs. That is, we assess the number of times an API consumer will be prompted to renegotiate an SLA due to the changes that occur in the cloud platform, and the time duration between these renegotiation events.

To enable this, we deploy Cerebro's cloud SDK monitoring agent in the Google App Engine cloud and benchmark the cloud SDK operations every 60 seconds for 112 days. We then use Cerebro to make SLA predictions (95th percentile) for the following set of open source web applications.

- *StudentInfo*: RESTful (JAX-RS) application for managing students of a class (adding, removing, and listing student information).
- *ServerHealth*: Monitors, computes, and reports statistics for server uptime for a given web URL.
- *StockTrader*: A stock trading application that provides APIs for adding users, registering companies, buying and selling stocks among users.
- *Rooms*: A hotel booking application with APIs for registering hotels and querying available rooms.

Cerebro analyzes the benchmarking results collected by the cloud SDK monitor and generates sequences of SLA predictions for the web APIs of each application. Each prediction sequence is a time series that spans the duration in which the cloud SDK monitor was active in the cloud. Each prediction is timestamped. Therefore given any timestamp that falls within the 112 day period of the experiment, we can find an SLA prediction that is closest to it. Further, we associate each prediction with an integer value (C_w) which indicates the consecutive number of SLA violations that should be observed, before we may consider the prediction to be invalid.

We also estimate the actual web API response times for the above four applications. This is done by simply summing up the benchmarking data gathered by the cloud SDK monitor. Again, we assume that the time spent on non cloud SDK operations is negligible. For example, consider a web API that executes the cloud SDK operations O_1, O_2 and O_1 in that order. Now suppose the cloud SDK monitor has gathered following benchmarking results for O_1 and O_2:

- O_1: $[t_1 : x_1,\ t_2 : x_2,\ t_3 : x_3 ...]$
- O_2: $[t_1 : y_1,\ t_2 : y_2,\ t_3 : y_3 ...]$

Here t_i are timestamps at which the benchmark operations were performed. x_i and y_i are execution times of the two SDK operations measured in milliseconds. Given this benchmarking data, we can calculate the time series of actual response time of the API as follows:

$[t_1 : 2x_1 + y_1,\ t_2 : 2x_2 + y_2,\ t_3 : 2x_3 + y_3 ...]$

The coefficient 2 that appears with each x_i term accounts for the fact that our web API invokes O_1 twice. In this manner, we can combine the static analysis results of Cerebro with the cloud SDK benchmarking data to obtain a time series of estimated actual response times for all web APIs in our sample applications.

Having obtained a time series of SLA predictions (T_p) and a time series of actual response times (T_a) for each web API, we perform the following computation. From T_p we pick a pair $< s_0, t_0 >$, where s_0 is a predicted SLA value and t_0 is the timestamp associated with it. Then starting from t_0, we scan the time series T_a to detect the earliest point in time at which we can consider the predicted SLA value s_0 as invalid. This is done by comparing s_0 against each entry in T_a that has a timestamp greater than or equal to t_0, until we see C_w consecutive entries that are larger than s_0. Here C_w is the rare event threshold computed by Cerebro when making SLA predictions. Having found such an SLA invalidation event at time t', we record the duration $t' - t_0$ (i.e. the SLA validity period), and increment the counter *invalidations*, which starts from 0. Then we pick the pair $< s_1, t_1 >$ from T_p where t_1 is the smallest timestamp greater than or equal to t', and s_1 is the predicted SLA value at that timestamp. Then we scan T_a starting from t_1, until we detect the next SLA invalidation (for s_1). We repeat this process until we exhaust either T_p or T_a. At the end of this computation we have a distribution of SLA validity periods, and the counter *invalidations* indicates the number of SLA invalidations we encountered in the process.

This experimental process simulates how a single API consumer (re-)negotiates SLAs. Selecting the first pair of values $< s_0, t_0 >$ represents the API consumer negotiating the SLA for the first time (i.e. at API subscription). When this SLA becomes invalid, the API consumer renegotiates for a new SLA, which is represented by the selection of the pair $< s_1, t_1 >$. Therefore, when the simulation reaches the end of the time series, we can determine how many times the API consumer had to renegotiate the SLA (given by *invalidations*). The recorded SLA validity periods give an indication of the time between these renegotiation events.

For a given web API we perform the above simulation many times, using each entry in T_p as a starting point. That is, in each run we change our selection of $< s_0, t_0 >$ to be a

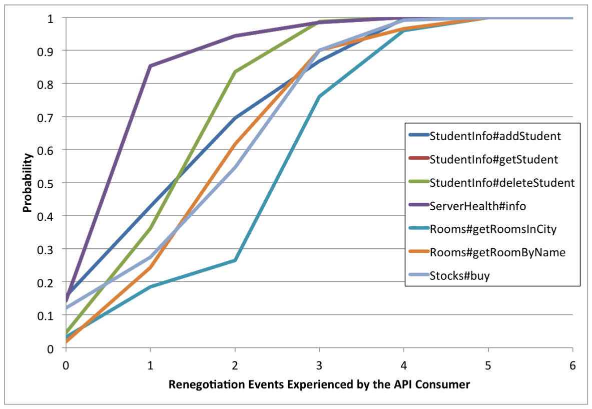

Figure 2. CDF of the number of renegotiation events faced by API consumers.

different entry in T_p. This way, for a time series comprised of n entries, we can run the simulation $n-1$ times, discarding the last entry. We can assume that each simulation run corresponds to a different API consumer. Therefore, at the end of a complete execution of the experiment we have the SLA renegotiation counts for many different API consumers, and the empirical SLA validity period distributions for each of them.

The smallest n we encountered in all our experiments was 125805. That is, we repeatedly simulated each web API SLA trace for at least 125804 API consumers. Similarly, the largest number of API consumers we performed the simulation for is 145130.

IV. Results

We next present the experimental results obtained using this methodology. We analyze the number of SLA renegotiations performed by each API consumer during the 112 day period of the experiment, and calculate a set of cumulative distribution functions (CDF). These CDFs describe the probability of finding an API consumer that experienced a given number of renegotiation events. Figure 2 presents the CDFs. We use the convention *ApplicationName#Operation* to label individual web API operations.

According to Figure 2, the largest number of SLA renegotiations experienced by any user is 6. This is with regard to the StudentInfo#addStudent operation. Across all web APIs, at least 96% of the API consumers experience no more than 4 renegotiation events during the period of 112 days. Further, at least 76% of the API consumers see no more than 3 SLA renegotiations. These statistics indicate that SLAs predicted by Cerebro for Google App Engine are fairly stable over time, and renegotiation is required only rarely. From an API consumer's perspective this is a highly desirable property, since it reduces the frequency and the overhead of SLA renegotiation.

Next we analyze the time duration between SLA renegotiation events. For this we combine the SLA validity

Operation	5^{th}	Mean	95^{th}
StudentInfo#getStudent	12.97	631.24	1911.19
StudentInfo#deleteStudent	7.65	472.07	2031.59
StudentInfo#addStudent	0.05	458.24	1711.08
ServerHealth#info	12.96	630.01	1911.19
Rooms#getRoomByName	8.48	345.13	1096.53
Rooms#getRoomsInCity	20.56	296.44	1143.45
Stocks#buy	8.46	411.75	815.5

Table I
Prediction validity period distributions (in hours). 5^{th} and 95^{th} columns represent the 5th and 95th percentiles of the distributions respectively.

periods computed for different API consumers into a single statistical distribution. Table I shows the 5th percentile, mean, and 95th percentile of these combined distributions.

The smallest mean SLA validity period observed in our experiments is 296.44 hours (12.35 days). This value is given by the Rooms#getRoomsInCity operation. This implies that on average, API consumers do not have to renegotiate Cerebro-predicted SLAs for at least 12.35 days. Similarly, we observed the largest mean SLA validity period of 26.3 days with the StudentInfo#getStudent operation. The smallest 5th percentile value of 0.05 hours is shown by the StudentInfo#addStudent operation, but this appears to be a special case compared to the other web API operations. The second smallest 5th percentile value of 7.65 hours is shown by the StudentInfo#deleteStudent operation. Therefore, ignoring the StudentInfo#addStudent operation, API consumers observe SLA validity periods longer than 7.65 hours at least 95% of the time. That is, the time between SLA renegotiations is greater than 7.65 hours at least 95% of the time.

To reduce the number of renegotiations further, we observe that we can exploit the SLA change events in which the difference between an invalidated SLA and a new SLA is small. In such cases, it is of little use to renegotiate a new SLA, and API consumers may be content to continue with the old SLA. To incorporate this behavior into Cerebro (and our simulation process), we introduce threshold value *sla_delta_threshold* into the process. This parameter takes a percentage value that represents the minimum acceptable percentage difference between the old and new SLA values before renegotiation. If the percentage difference between the two SLA values is below this threshold, we do not record the SLA validity period, nor increment the count of the SLA invalidations. That is, we do not consider such cases as renegotiation events. We simply carry on with the old SLA value until we come across an invalidation event with a percentage difference that exceeds the threshold. Note that our previous experiments are a special case of thresholding for which *sla_delta_threshold* is 0.

Next we evaluate the sensitivity of our results to *sla_delta_threshold*. Figure 3 shows the resulting CDFs of

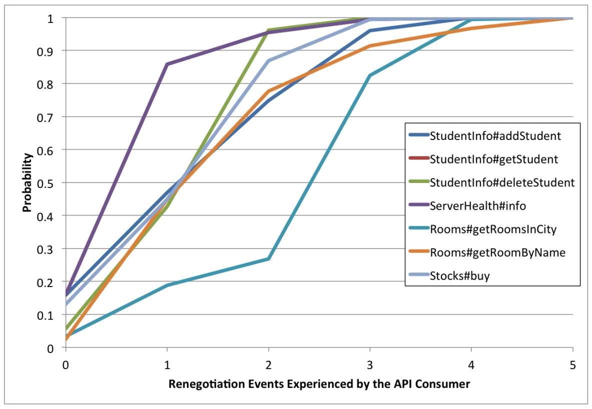

Figure 3. CDF of the number of renegotiation events faced by API consumers, when *sla_delta_threshold* = 10%

Operation	5^{th}	Mean	95^{th}
StudentInfo#getStudent	19.93	644.58	1911.19
StudentInfo#deleteStudent	7.93	512.52	2031.59
StudentInfo#addStudent	0.05	491.68	1711.08
ServerHealth#info	19.91	643.33	1911.19
Rooms#getRoomByName	8.48	392.01	1096.53
Rooms#getRoomsInCity	21.82	304.97	1143.45
Stocks#buy	7.41	510.31	1277.7

Table II
PREDICTION VALIDITY PERIOD DISTRIBUTIONS (IN HOURS) WHEN *sla_delta_threshold* = 10%. 5^{th} AND 95^{th} COLUMNS REPRESENT THE 5TH AND 95TH PERCENTILES OF THE DISTRIBUTIONS RESPECTIVELY.

per-user renegotiation count when the threshold is 10%. That is, Cerebro does not prompt the API consumer to renegotiate an SLA, unless the new SLA is at least 10% off from the old one. In this case, the maximum number of renegotiation events drops from 6 to 5. Also most of the probabilities shift slightly upwards. For instance, now more than 82% of the users see 3 or less renegotiation events (as opposed to 76%).

Table II shows the SLA validity period distributions computed when *sla_delta_threshold* is 10%. Here, as expected most of the mean and 5th percentile values have increased slightly from their original values. The smallest mean value recorded in the table is 304.97 hours. We have also considered a *sla_delta_threshold* value of 20%. This change introduces only small shifts in the probability values of the CDFs (more than 84% of the users see 3 or less renegotiations), and the maximum number of renegotiations remains at 5.

In summary, we find that the performance SLAs predicted by Cerebro for the Google App Engine cloud environment are stable over time. That is, the predictions are valid for long periods of time, and API consumers need not renegotiate the SLAs often. In our experiment spanning over a period of 112 days, the maximum number of renegotiations a user had to undergo was 6. More than 76% of the users experienced only 3 or less renegotiations. We can further reduce the number of SLA renegotiations per API consumer by introducing a threshold for the minimum applicable percentage SLA change. This helps to eliminate the cases where an old SLA has been marked as invalid by our statistical model for detecting SLA invalidations, but the new SLA predicted by Cerebro is not very different from the old one. However, the effect of this parameter starts to diminish as we increase its value. In our experiments, we observe the best results for a threshold of 10%. Using a value of 20% does not achieve significantly better results.

V. Related Work

SLA management on service-oriented systems and cloud systems has been studied in some depth previously. Much of this existing work has focused on issues such as SLA monitoring [11], [12], [13], [14] and SLA modeling [15], [16], [17]. In our work, we automatically identify the SLAs that can be defined and maintained for a given web API by using a combination of static analysis, cloud platform monitoring and time series analysis.

In PROSDIN [18], a proactive service discovery and negotiation framework, the SLA negotiation occurs during the service discovery phase. This is similar to how Cerebro establishes an initial SLA with an API consumer, when the consumer subscribes to an API. PROSDIN also establishes a fixed SLA validity period upon negotiation, and triggers an SLA renegotiation when this time period has elapsed. Cerebro on the other hand continuously monitors the cloud platform, and periodically re-evaluates the response time SLAs of web APIs to determine when a re-negotiation is needed. Similarly, researchers have investigated the notions of SLA brokering [19], and the automatic SLA negotiation between intelligent agents [20], ideas that can complement the simple SLA negotiation model of Cerebro to make it more powerful and flexible.

Meryn [21] is an SLA-driven PaaS system that attempts to maximize cloud provider profit, while providing the best possible quality of service to the cloud users. It supports SLA negotiation at application deployment, and SLA monitoring to detect violations. However, it does not automatically determine what SLAs are feasible or address SLA renegotiation, and employs a policy-based mechanism coupled with a penalty cost charged against the cloud provider to handle SLA violations. Also, Meryn formulates SLAs in terms of the computing resources (CPU, memory, storage etc.) allocated to applications. It assumes a batch processing environment where the execution time of an application is approximated based on a detailed description of the application provided by the developer. In contrast, Cerebro handles SLAs for interactive web applications. It predicts the response time of applications using static analysis, without any input from the application developer. Cerebro also supports automatic SLA renegotiation, with possible room for economic incentives.

Iosup et al showed via empirical analysis, that production cloud platforms like Google App Engine and AWS regularly undergo performance variations, thus impacting the response time of the applications deployed in such cloud platforms [5]. Some of these cloud platforms even exhibit temporal patterns in their performance variations (weekly, monthly, annual or seasonal). Cerebro and the associated API performance forecasting model acknowledge this fact, and periodically re-evaluate the predicted response time upper bounds. It detects when a previously predicted upper bound becomes invalid, and prompts the API clients to renegotiate their SLAs accordingly. Indeed, one of Cerebro's strength's is its ability to detect change points in the input time series data (periodically collected cloud SDK benchmark results), and generate up-to-date predictions that are not affected by old obsolete observations that were gathered prior to a change point.

There has also been prior work in the area of predicting SLA violations [22], [23], [24]. These systems take an existing SLA and historical performance data of a service, and predict when the service might violate the given SLA in the future. Cerebro's notion of prediction validity period has some commonalities with this concept. In fact, Cerebro can make use of such a method to determine the frequency at which it should re-evaluate the predicted SLAs.

VI. Conclusion

Web APIs impact the correctness and performance of the applications that depend on them. However, most web APIs do not provide SLAs with regard to such properties, making it difficult to reason about the performance of the applications that use web APIs. Cerebro attempts to solve this problem for web APIs developed for PaaS clouds, by automatically predicting response-time SLAs for them.

In this work we analyze the validity period of the SLAs predicted by Cerebro. We present a simple SLA negotiation model, and a conservative statistical approach for detecting when a predicted SLA has become invalid. We evaluate our methods on the Google App Engine public cloud using empirical testing and simulations. We find that on App Engine, Cerebro predictions are valid for at least 12 days on average. We also find that API consumers do not have to renegotiate SLAs often, and the maximum number of times an API consumer must renegotiate an SLA over a period of 112 days is six. We also present and evaluate a threshold-based mechanism to eliminate the SLA renegotiations where the old and new SLA values are very close to each other. This optimization further increases the average SLA validity period, and reduces the number of SLA renegotiations per API consumer. Overall, this work shows that automatic definition of response-time SLAs for web APIs is practically viable in real world cloud settings, and API consumer timeframes.

Acknowledgment

This work is funded in part by NSF (CCF-1539586, CNS-1218808, CNS-0905237, ACI-0751315), NIH (1R01EB014877-01), and the California Energy Commission (PON-14-304).

References

[1] "ProgrammableWeb," http://www.programmableweb.com [Accessed March 2015].

[2] A. Keller and H. Ludwig, "The wsla framework: Specifying and monitoring service level agreements for web services," *J. Netw. Syst. Manage.*, vol. 11, no. 1, Mar. 2003.

[3] H. Jayathilaka, C. Krintz, and R. Wolski, "Response time service level agreements for cloud-hosted web applications," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, 2015.

[4] C. Krintz, "The appscale cloud platform: Enabling portable, scalable web application deployment," *Internet Computing, IEEE*, vol. 17, no. 2, pp. 72–75, March 2013.

[5] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011.

[6] https://cloud.google.com/appengine/docs/quotas [Accessed March 2015].

[7] http://azure.microsoft.com/en-us/documentation/articles/azure-subscription-service-limits/#cloud-service-limits [Accessed March 2015].

[8] "Google app engine java sandbox," 2015, "https://cloud.google.com/appengine/docs/java/#Java_The_sandbox" [Accessed March 2015].

[9] S. Bygde, "Static wcet analysis based on abstract interpretation and counting of elements," Ph.D. dissertation, Mälardalen University, 2010.

[10] D. Nurmi, J. Brevik, and R. Wolski, "QBETS: Queue Bounds Estimation from Time Series," in *International Conference on Job Scheduling Strategies for Parallel Processing*, 2008.

[11] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Comprehensive QoS Monitoring of Web Services and Event-based SLA Violation Detection," in *International Workshop on Middleware for Service Oriented Computing*, 2009.

[12] A. K. Tripathy and M. R. Patra, "Modeling and Monitoring SLA for Service Based Systems," in *International Conference on Intelligent Semantic Web-Services and Applications*, 2011.

[13] F. Raimondi, J. Skene, and W. Emmerich, "Efficient Online Monitoring of Web-service SLAs," in *ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2008.

[14] A. Bertolino, G. De Angelis, A. Sabetta, and S. Elbaum, "Scaling Up SLA Monitoring in Pervasive Environments," in *Workshop on Engineering of Software Services for Pervasive Environments*, 2007.

[15] T. Chau, V. Muthusamy, H.-A. Jacobsen, E. Litani, A. Chan, and P. Coulthard, "Automating SLA Modeling," in *Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, 2008.

[16] K. Stamou, V. Kantere, J.-H. Morin, and M. Georgiou, "A SLA Graph Model for Data Services," in *International Workshop on Cloud Data Management*, 2013.

[17] J. Skene, D. D. Lamanna, and W. Emmerich, "Precise Service Level Agreements," in *International Conference on Software Engineering*, 2004.

[18] K. Mahbub and G. Spanoudakis, "Proactive SLA Negotiation for Service Based Systems: Initial Implementation and Evaluation Experience," in *IEEE International Conference on Services Computing*, 2011.

[19] L. Wu, S. Garg, R. Buyya, C. Chen, and S. Versteeg, "Automated SLA Negotiation Framework for Cloud Computing," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2013.

[20] E. Yaqub, R. Yahyapour, P. Wieder, C. Kotsokalis, K. Lu, and A. I. Jehangiri, "Optimal negotiation of service level agreements for cloud-based services through autonomous agents," in *IEEE International Conference on Services Computing*, 2014.

[21] D. Dib, N. Parlavantzas, and C. Morin, "Meryn: Open, SLA-driven, Cloud Bursting PaaS," in *Proceedings of the First ACM Workshop on Optimization Techniques for Resources Management in Clouds*, 2013.

[22] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime Prediction of Service Level Agreement Violations for Composite Services," in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, ser. Lecture Notes in Computer Science, A. Dan, F. Gittler, and F. Toumani, Eds. Springer Berlin Heidelberg, 2010, vol. 6275, pp. 176–186.

[23] B. Tang and M. Tang, "Bayesian Model-Based Prediction of Service Level Agreement Violations for Cloud Services," in *Theoretical Aspects of Software Engineering Conference (TASE)*, 2014.

[24] S. Duan and S. Babu, "Proactive Identification of Performance Problems," in *ACM SIGMOD International Conference on Management of Data*, 2006.