OptCon: a Flexible Situation-Aware Framework for Consistency Management

Subhajit Sidhanta
Louisiana State University

Wojciech Golab
University of Waterloo

Supratik Mukhopadhyay
Louisiana State University

Abstract

Users of distributed data stores that employ quorum-based replication are burdened with the choice of a client-side consistency level for each storage operation. The matching choice is difficult to reason about as it depends on the application requirements, as well as the network latencies and processing delays at a given time. We present OptCon: a novel predictive framework that can automate this choice given a user-specified combination of thresholds in the form of an SLA. OptCon predicts the strongest client side consistency level that can be applied under the current state of the network and the system, while satisfying the SLA with respect to three metrics: consistency, latency, and packet retransmission rate. As we show experimentally, OptCon provides an intelligent combination of strong consistency in certain feasible scenarios as well as supporting weaker forms of consistency in other cases based on the demand of the situation. In particular, we demonstrate using the RuBBOS benchmark application that OptCon is as effective as an optimal combination of fixed consistency settings.

1 Introduction

Many distributed data storage systems use quorum-based replication to enable fault-tolerance and improve availability. Replication is essential to deal with variations in load, and growing number of parallel requests [15]. Replication comes with the added cost of maintaining *consistency* across replicas through communication between them [2, 38]. As it is impossible to achieve all the three objectives of Brewer's CAP theorem (i.e., consistency, availability, and partition-tolerance [9, 16, 3]), large scale distributed storage systems overcome this limitation by offering relaxed consistency guarantees. Commercial and open-source storage systems [22, 39, 30, 32, 37, 12] either offer "one size fits all" consistency guarantees that do not take into account the client requirements, system load, and the network statistics, or allow the user to manually choose an appropriate consistency level a priori. Storage systems like Google's Spanner [11] are specifically designed to offer strong consistency regardless of the needs of the user and the conditions of the system or the network. Cassandra [22] allows manual tuning by users/clients to obtain customized consistency guarantees. Users need a thorough understanding of the impact of a specific consistency level for a given set of operations, the background synchronization tasks, and the network statistics on the client-side performance to decide upon the correct consistency setting for a given use case. Thus, neither commercial nor open-source distributed datastores customize the consistency settings automatically according to each use case, system, and network conditions—all one can do is specify the consistency setting manually prior to an operation. A particular consistency level, however, may not be appropriate for all use cases: real time operations would require strong consistency, while batch or transaction operations call for weaker consistency settings [40].

Recent research has considered techniques for specifying application requirements, including consistency, in a fine-grained manner using service level agreements (SLAs). Terry et al. introduce Pileus, which can be configured to provide session guarantees such as monotonic reads, as well as bounded staleness [40]. The requirements are specified as a sequence of subSLAs (i.e., rows in an SLA) ordered by a user-defined utility metric, which the system tries to maximize. Tuba is a geo-replicated system that supports similar SLAs and is able to automatically reconfigure it set of replicas in response to changing client locations or request rates [5]. Pileus and Tuba are both tightly coupled to proprietary data stores whereas our goal is to provide a framework that can be applied to any existing distributed data store.

Several other systems have been proposed to simplify consistency tuning and could be applied on top of existing eventually consistent data stores. Bailis et al. use

Monte Carlo methods for calculating the probability of a stale read occurring at a fixed time t following a write [6]. Their mathematical model is based upon the simplifying assumption that writes do not execute concurrently with other operations, and for that reason it is not clear how one would compute t from a real workload. Rahman et al. provide a probabilistic consistency-latency tradeoff mainly by delaying reads artificially at clients [34]. If the storage system is deployed in a single data center with a fast network then this technique provides a poor trade-off because all reads pay a latency penalty for the sake of a small fraction of reads that would otherwise return a stale value. In comparison, majority quorums guarantee strong consistency deterministically with only slightly higher latency compared to eventual consistency.

2 Contributions

This paper presents OptCon, a novel predictive framework that can automate the choice of client-side consistency settings to achieve a user-specified combination of performance thresholds in the form of an SLA. OptCon predicts the strongest client-side consistency level that can be applied under the current state of the network and the system, while satisfying the SLA with respect to three metrics: consistency, latency, and packet retransmission rate. In contrast to Pileus and Tuba, our framework is not tied to any specific storage back end, and can be applied on top of any system that provides configurable client-side consistency settings on a per-operation basis.

Our SLAs are structured as a sequence of subSLAs similarly to Terry et al. [40], as illustrated in Table 1. A consistency level is *matching* with respect to a subSLA if the performance metrics—latency, staleness, and packet retransmission rate—fall within the thresholds specified in the subSLA when that particular consistency level is used. Users can specify different threshold values for each of the performance metrics, according to the requirements of the particular use case and particular application category. For applications like web search, where latency is the priority, users can use the subSLA to specify a stringent threshold for latency. For example, for a particular operation, a user may state that response time for any search query must not exceed 10 milliseconds. For batch processing operations where latency has very low priority, like addition of new videos in an online video streaming database (like Netflix), or overnight database migration activities, users may specify that the write operation is acceptable as long as the latency or response time does not exceed a threshold of 12 hours, for example. OptCon enables data stores to opportunistically attempt to apply the strongest consistency level that meets the performance thresholds corresponding to a subSLA. In case of multiple consistency options satisfying the given thresholds, OptCon applies the strongest of the candidate client side consistency levels. This ensures that the user gets the best possible tradeoff of latency, staleness, and packet retransmission rate for a particular use case under a given network state, without violating the thresholds specified in the subSLA.

Table 1: Example of subSLA

Rank	Latency	Staleness	Retransmission
1	≤60ms	≤100ms	≤10/s
2	≤25ms	≤60ms	≤20/s
3	≤10ms	≤30ms	≤10/s

The paper makes the following contributions: (1) We describe the high-level architecture of OptCon, our system for predicting the matching client-side consistency level, and discuss several alternatives for its internal design: mathematical modeling using linear regression, as well as machine learning using a decision tree classifier and a Beysian network. (2) We evaluate the effectiveness of the framework over changing workloads and different subSLAs, spanning over a range of latency and staleness thresholds. (3) We demonstrate using the RuBBOS benchmark application that our framework is as effective as an optimal combination of fixed consistency settings. We define two metrics M and M_C (refer to Section 5) to quantify the performance of the framework with respect to given subSLAs in these experiments.

3 Overview of OptCon

Our implementation of OptCon (see Figure 1) consists of 1) a Logger module that gathers the performance metrics and system parameters using the Java Management Extensions (JMX) Interface provided by the storage system, 2) a Learner module that runs the Machine Learning based Training Data in the experiment stage, and 3) a Client module that predicts the matching consistency level, and performs the requested operation by calling the query client.

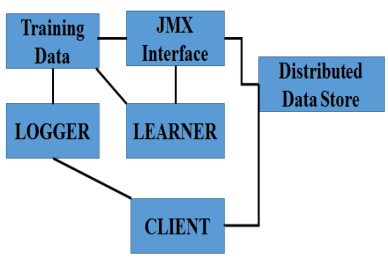

Figure 1: The architecture of OptCon

The JMX Interface module is an extension over YCSB

0.1.4, running on top of a Distributed DataStore, collecting various parameters and statistics about the state of an operation. The Logger module collates the statistics into a Training data file. In the absence of a closed form mathematical model formally relating the parameters—client side consistency, observed latency, and staleness—we use machine learning to learn a model from historic data. The Learner module builds a classifier from the training Data, classifying each set of parameters according to the matching client side consistency, and the Client module predicts the strongest consistency level that is possible, under a given set of thresholds specified in the subSLAs. OptCon is meant to act as a wrapper over any existing distributed storage system (with minor modifications in the wrapper code), allowing users to specify bounds to latency, observed consistency (i.e., staleness), and packet retransmission rate in subSLA format.

We have chosen operation latency L, TCP packet retransmission rate R (retransmission rate reflects the network behavior in case of failure), and average staleness S as the set of performance metrics to be optimized, under a particular set of controlled parameters (tuning knobs): thread count, i.e., number of the client threads, the packet count, and read-write proportions. Staleness is the difference in state/value of replicas, which depends on the replication factor and topology, and is measured in terms of the Γ metric [18]. The Learner module learns a model from the Training Data, consisting of the above variables collected by the JMX Interface and collated by the Logger module, and predicts matching client side consistency settings under a given subSLA.

The Γ metric for staleness is based upon Lamport's atomicity property [24], which states that operations appear to take effect in some total order that reflects their "happens before" relation in the following sense: if operation A finishes before operation B starts then A must appear to take effect before B. We say that a trace of operations recorded by the logger is Γ-atomic if it is atomic in Lamport's sense, or becomes atomic after each operation in the trace is transformed by decreasing its starting time by $\Gamma/2$ time units, and increasing its finish time by $\Gamma/2$ time units. In this context the start and finish times are shifted in a mathematical sense for the purpose of analyzing a trace, and do not imply injection of artificial delays; the behavior of the storage system is unaffected. The Γ metric can be defined at various granularities, including a *per-key* Γ *score* that quantifies the degree of staleness incurred by operations applied to a particular key [18]. We adopt as our measure of staleness an *average* Γ *score* defined as follows:

$$\frac{\sum \text{per-key } \Gamma \text{ scores}}{\text{total \# of keys accessed in the trace}}$$

For the Learner module we exploit three separate machine learning approaches [21, 46, 36] —linear regression, decision tree, and Bayesian network [14]. We discuss the performance of each of the techniques in later sections. We started with the linear regression technique in an attempt to come up with a simple mathematical model in the form of linear equations, with the performance metrics as dependent variables and the controlled parameters as independent variables. We use minimization of the Root Mean Square Error (RMSE), to obtain the underlying parametric model representing the dependency of the optimization parameters. We minimize the chances of overfitting the regression curve by using varying workloads and artificially introduced traffic congestion. As regards machine learning techniques, we chose the decision tree because of its simplicity, speed, and because we can cast our problem as that of classifying the data in terms of the strongest consistency level for the rows that satisfy the threshold values given in the subSLA. Particularly we choose an error pruning decision tree, thus mitigating issues with overfitting [14]. We also use a Bayesian network to model the dependencies of the dependent the variables on the controlled parameters. In case OptCon predicts of multiple client side consistency options satisfying a given subSLA under a given network state, the strongest candidate client side consistency levels can be chosen, to provide the user with the best possible tradeoff of the latency, staleness, and packet retransmission rate. In the next subsections we discuss each of the three techniques underlying the learning module of OptCon.

3.1 Prediction of Matching Consistency using Linear Regression Approach

We used Linear Regression to obtain the dependency relation between the optimized parameters (i.e., dependent variables) and the controlled parameters with separate objective functions for each of the dependent variables. The respective objective functions are computed from the data, and are converted to mathematical representations of the form

$$L = X_{11} \times C + X_{12} \times RW + X_{13} \times Tc + X_{14} \times P \quad (1)$$

$$S = X_{31} \times C + X_{32} \times RW + X_{33} \times Tc + X_{34} \times P \quad (2)$$

$$R = X_{41} \times C + X_{42} \times RW + X_{43} \times Tc + X_{44} \times P \quad (3)$$

where L, S and R are the dependent variables described earlier, C is the client specified consistency level for both read and write operations, RW is the read-write proportion, Tc is the client thread count, and P is the packet count (i.e., the number of packets captured while performing the given operation). The consistency level C in its raw form is a string representing the consistency settings applied for each operation but we convert this cat-

egorical attribute to a numerical form by assigning each consistency level a numerical value. This allows us to integrate the consistency level into the regression model.

Our target is to select the strongest consistency level that produces values for the dependent variables within the thresholds specified in the subSLAs, given a combination of values for the input parameters Tc, P and RW. We model each L, S and R individually and fit linear equations for each dependent variable, which we then use to predict whether a particular subSLAs can be satisfied given the values of Tc, P and RW representing the workload and network state as well as a particular client-side consistency level C. For each candidate consistency level C, the predicted value for each of the dependent variables is compared with the corresponding threshold value specified in the subSLA. Given multiple values of C that satisfy all the subSLA we choose the strongest consistency level.

To understand the behavior of the regression model we perform an ANOVA analysis [14] and determine the significance of each parameter in the model equations. Statistical significance is determined by the F-ratios, and probabilistic significance (i.e., estimated probability of obtaining observed results given the null hypothesis is true) is denoted by the p values.

3.2 Learning Matching Consistency with Decision Tree Approach

This approach uses the Decision Tree technique for predicting the matching consistency level, and Exhaustive search for labelling the training data by the strongest consistency level for each set of RW, Tc and P values such that the dependent variables are within the respective threshold limits. The above technique searches the training data set, to organize the data into groups against each tuple $\langle RW, Tc, P\rangle$, obtains the row k with strongest consistency level C, and labels respective group with the corresponding consistency level C_k associated with the row k. The above labelled data set is then fed into a machine learning module as training data - in our case we have chosen decision tree. With the above objective functions (refer to section 3.1) and the labelled training data obtained from exhaustive search, we can train a classifier to learn the matching consistency level for a set of measured read-write proportion, thread count and packet count. In the prediction phase, we predict the consistency level C that is the maximum consistency setting possible, under given values for the independent variables.

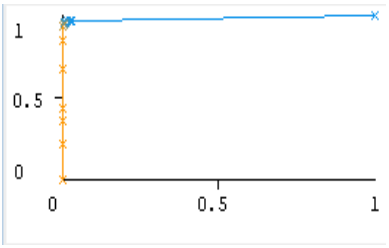

Figure 2: ROC curve generated using Decision Tree: AUC=0.9783

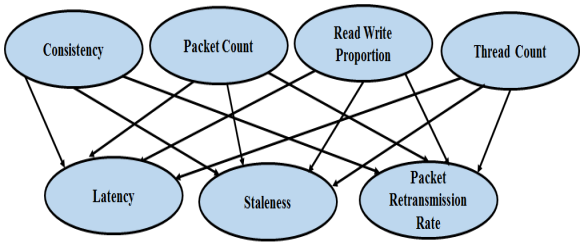

Figure 3: The belief network graph generated from the Bayesian Network approach

3.3 Inference of Matching Consistency level with Bayesian Network Method

With the Bayesian approach, we assume a prior distribution for each of RW, Tc, P, and C. The posterior distributions for L, S, and R are conditionally dependent on the joint distribution of $\langle RW, Tc, P, C\rangle$. We obtain the dependency relation of the variables from the linear equations obtained from the Linear Regression approach. The resultant dependency graph is then input to the Bayesian Network for generating the model with the given variables. Figure 3 illustrates such a graph. As before, we choose consistency levels for which the values of the dependent variables fall within the ranges stipulated by the threshold values in the subSLAs. We assume the attributes RW, Tc, P, and C comprising the knob X, given by the set of input variables, are conditionally independent on each other. We assume that the target features L, T, S, and R are also conditionally independent (although actually L and S are dependent on each other). The above assumptions allow us to apply the simplistic (yet powerful and appropriate) naive Bayes model in calculating the maximum a posterior value for C that maximizes the likelihood. Our target is to model the causality among the dependent and independent variables as a simple Bayesian Network, and compare the results thus obtained with our other approaches, for the sake of exploring and evaluating different methods.

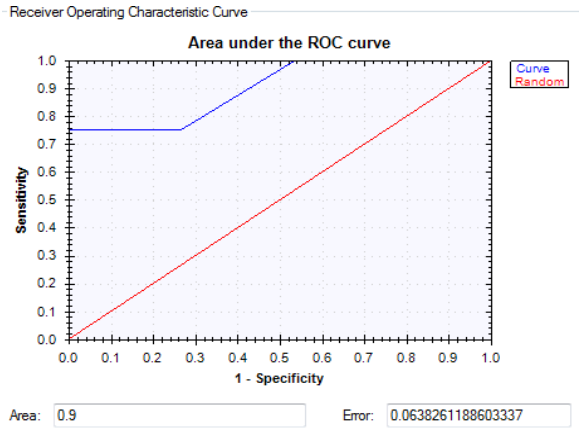

Figure 4: ROC curve generated using Bayes Net

4 Implementation and Evaluation

Our implementation targets Cassandra as the key value store, and uses YCSB as a client for performing read/write operations on Cassandra. Cassandra is an open source key value based distributed data store, and is widely used in the industry as well as academic world for its scalability, partition tolerance, and availability. However, OptCon has been built as a Java-based wrapper, following a loosely-coupled architecture, and can be readily integrated with any distributed storage system with minimum rework. The users have the choice of either having OptCon as a layer over the underlying data store or manually specifying the client side consistency settings while performing an operation.

4.1 Experimental Setup

We have run our experiments on a test bed consisting of a cluster of 20 Amazon ec2 micro instances, running Cassandra over Ubuntu 13.10 LTS operating system. We have developed the Logging module as an extension of the widely recognised YCSB benchmark tool (version 0.1.4) running on top of Cassandra 2.1.0. For each operation, it writes the key id, start time, finish time, the operation type (read/write/scan/update), for calculation of the (Γ)) metric. It collects the various parameters and measures about Cassandra's state from various sources like the Java Management Bean Extension (JMX) interface of Cassandra, the OperatingSystemMXBean API exposed by the JVM, the Thread Count measure collected from the operating system, and the packet count from libpcap API, as well as latency and read-write proportions obtained from the underlying YCSB core. The above generated dataset is stored in the filesystem for later analysis phase.

For the Linear Regression approach, we preprocessed the generated dataset file and used Matlab 2013b's statistical toolbox to build the model, represented as separate linear equations for each of the dependent attributes. For each dependent variable, OptCon takes numeric values as thresholds from the user in the form of subSLA rows - each row has a rank attached marking its preferential order. The ranks in the subSLA are used as preference values for the user to try out consistency levels in an ascending order of ranks in case of failure scenarios. For each set of given values for the independent parameters, the dependent variables are evaluated against different consistency levels, and we select the strongest consistency level that produces output parameters within specified threshold limits. We rate consistency levels as per nature the operation (i.e., read/write) being performed; for a write operation (i.e., write/update) READ-ONE/WRITE-ALL is considered stronger than READ-ALL/WRITE-ONE, while for read operation (i.e., read/scan) it is the other way round.

For the Decision Tree approach we used J48, the decision tree API of Weka with reduced error pruning. Weka is an open source machine learning suite implementing a host of machine learning algorithms. During training phase, we run a set of YCSB workloads on top of the Cassandra cluster with different consistency settings, repetitively on the same set of keys, with a replication factor of 5. The throughput, latency and other parameter values obtained via the JMXInterface wrapper of Cassandra are written to the log files. The client retrieves the classifier generated by the Training module from the file system, and obtains the predicted consistency level for a current realtime operation. The total delay recorded for the client operation ranges between 5 to 10 ms, with the call to the classifier ranging from 3-5 ms on an average.

For the Bayesian approach, we use Infer.net, an open source machine learning library under the .NET framework, managed by Microsoft Research, to build the Bayesian Network which is used to build the classifier. Then the above model is used to make the matching consistency level predictions for the test dataset. Subsequent analysis, i.e., ROC curve generation, etc. are carried upon the spreadsheet for visualising the predictions. The source code for OptCon implementation can be found in the github repositories : `https://github.com/ssidhanta/YCSBpatchpredictconsistency/`, `https://github.com/ssidhanta/TrainingModelGenerator/`, and `https://github.com/ssidhanta/HectorCient/`.

4.2 Data Collection

We executed read/write operations on a Cassandra cluster with various types of core YCSB workloads and randomly selected consistency levels (predefined for Cas-

sandra 2.0.x), and logged in the various metrics (discussed in Section 3) for each operation. In this manner, we have generated a dataset of almost 25,600 rows - each row consisting of 10 parameters or features, namely latency, throughput, gamma metric (staleness), packet retransmission rate, thread count, packet count, consistency level, read-write proportion, cpu load, and operation type. We ran the experiments from multiple YCSB clients, in order to simulate the distributed nature of usual user operations, and the statistics were collected from multiple JMX clients, listening on different nodes across the network. Since the cluster where the operations took place across consists of multiple disjoint nodes connected by a network with lags/time delays in the individual system clocks - they are prone to measurement error due to clock skews in various nodes of the cluster. Without time synchronization among the various nodes, the calculated measures like latency will be rendered useless because of the clock skew. We have used the NTP (Network Time Protocol) protocol implementation from ntp.org to synchronize clocks on the servers to one designated cluster machine rather than relying on an external time source as that would lead to a clock skew of 10ms or worse in the Amazon EC2 environment.

4.3 Experimental Procedure

We have split the above dataset into training and test set - 2/3rd of the rows make the training data, while the rest is used for testing. With the Linear Regression approach, we have used the curve fitting toolbox of Matlab to obtain a set of linear equations, one for each of the dependent variables L, R, S constructed from the dataset. L is given in milliseconds, R in number of retransmissions/second, and S is measured in terms of milliseconds. Here we consider matching latency-consistency tradeoff, determining matching consistency level in order to minimize latency L, as per the demand of the use case. Studies like [13] suggest that the latency for standard web applications varies in the range of 75-140 ms. The Pileus system [40], which includes latency in subSLAs, uses 200 to 1000 ms as the latency threshold - thus any value between 100 and 1000ms can be a candidate for the threshold of latency in our case. The ANOVA analysis confirms that variables S and L has greater significance than variable R, since they have higher F statistics (refer to section 4.4). In the work on probabilistically bounded staleness [6], the t-visibility values recorded, in the Riak deployment at Yammer [19], range between 186 and 1364 ms - hence any value within that range might be used for staleness threshold. In [42] the authors state that the typical bounds for TCP retransmission rate is 200 to 400 ms. Retransmission rate has much lesser importance (hence magnitude) than both latency and staleness in the objective function - hence we must set the threshold for retransmission rate between one and 11. To sum up, the subSLAs are simulated as follows: random numbers generated uniformly from a range between one and 11 retransmission/second are assigned to Y_4 for R respectively. A positive value between 101 and 150 ms is assigned to the threshold Y_1 for variables L, and a value between 41 and 101 ms is assigned to Y_3 for S respectively. Default values for the threshold Y_1 is 150 ms and the default for Y_3 is 41 ms, corresponding to variables L and S respectively. The default value for Y_4 is 11 retransmission/second corresponding to variable R respectively. With each row in test data set, we try out each combination of independent variables against all possible consistency levels and determine (or in other words predict) the combination that gives the strongest consistency level. Thus, starting with the equations obtained with Linear Regression, we predict the strongest possible consistency levels under the given combination of input parameters.

In Decision Tree approach, we train the classifier over the above training dataset to learn the matching consistency levels for each values of the input parameters. We tested the system by running custom subSLA's over the test data - and compared the predicted consistency level with the matching consistency recorded in the last column of the dataset by the Exhaustive Search labelling mechanism.

With Bayesian Inference method, we use Infer.net to define Dirichlet priors for the independent variables; then we form the conditional posteriors for the dependent variables with a single child from four parents. We set each independent variable in the training dataset as the observed value for the priors and each dependent variable as observed value for the posteriors. Then we call upon the Infer method to learn the Bayesian network and derive the matching consistency levels for each combination of priors. Next we write the predicted consistency levels for each test data on an excel; subsequently we operate on the excel to obtain the ROC curve (refer to Figure 4).

4.4 Experimental Results: Discussion of ANOVA Table

We perform an analysis of variance test (ANOVA) on the fitted functions for each dependent variables (refer to Figure 2) and observed the following.

1) In the case of retransmission R, the F-ratio is small for all parameters (closer to one) indicating the null hypothesis tends to be true. The p-value for all input parameters are very high, equals to 1; thus all the dependent parameters have insignificant contribution to the response variable.

2) In case of latency L, the F-ratios are not quite high

and mostly close to 1, indicating considerable statistical significance. On the other hand, p values are considerably low indicating indicates that the chances of the null hypothesis being true is low. Hence L has moderate, i.e., not very high, significance among the variables in the objective function. Thus, the thresholds for L in the prediction criteria are moderately high in proportion to other parameters.

3) Finally, in the case of the function for staleness S, the F-ratios are much more closer to 1, and the p values for all input parameters are also much low indicating high probabilistic significance; hence the staleness function is given high significance as a dependent parameter.

Thus, linear regression suggests that the given objective functions are not worth considering for prediction of matching consistency. Moreover, the ROC curve follows the diagonal and has an area close to 0.5 - hence its prediction power is no more than random predictions. The problems with the Linear Regression approach prompts us to look to our Decision Tree approach, in hope for better results.

Table 2: ANOVA results for Linear Regression

ANOVA for Latency function					
Variable	**SumSq**	**DF**	**MeanSq**	**F**	**pValue**
C	4.11E+18	1	4.11E+18	7.0851	7.82E-03
RW	1.57E+20	1	1.57E+20	269.75	1.23E-57
P	7.26E+18	1	7.26E+18	12.503	0.00041377
Error	1.45E+21	2492	5.80E+17		

ANOVA for Staleness function					
Variable	**SumSq**	**DF**	**MeanSq**	**F**	**pValue**
C	7.04E+08	1	7.04E+08	1.1906	0.27523
RW	7.71E+09	1	7.71E+09	480.32	1.70E-105
P	9.71E+08	1	9.71E+08	3.6821	0.05501
Error	1.92E+12	2492	7.69E+08		

ANOVA for Retransmission function					
Variable	**SumSq**	**DF**	**MeanSq**	**F**	**pValue**
C	1.57E-26	1	1.57E-26	3.66E-15	1
RW	4.38E-25	1	4.38E-25	1.02E-13	1
P	4.31E-25	1	4.31E-25	1.01E-13	1
Error	1.07E-08	2492	4.29E-12		

4.5 Experimental Results: Discussion of Generated Curves

Figure 3 gives the confusion matrix for the Decision Tree method; the actual class labels are given along rows and predicted classes are given across the rows. From the confusion matrix, the ROC curve (or TPR vs FPR curve) Figure 2 is obtained. The area under the curve AUC, computed using Mann Whitney [28], is 0.9857, which is very close to 1 - proving the method to have reasonably high prediction power. Gini coefficient G_1 or the distance of the curve from the diagonal is $G_1 = 2AUC - 1 = 2 \times 0.9857 - 1 = 0.9714$.

The ideal region for ROC curve is the upper leftmost corner whereas it is the upper rightmost corner for the PR curve; hence the results are accurate enough. The ROC curve obtained from the Bayes network approach is given in Figure 4; the prediction points all fall above the random guess line (diagonal) - hence predictions are fairly accurate. The area under the curve AUC is 0.9, indicating it to be a sufficiently accurate predictor. The distance of prediction points from the diagonal given by the Gini coefficient G_1 is $G_1 = 2 \times 0.9 - 1 = 0.8$. But for Linear regression, the ROC curve follows the diagonal and AUC = 0.5 and G_1 = 0, and hence is no better than random chance.

Table 3: Confusion matrix generated using Decision Tree

a	**b**	**c,d,e,f**	**g**	**h**	**<– classified as**
6194	238	0	73	93	a = ANY
115	1045	0	2793	495	b = ONE
0	0	0	0	0	c = TWO
0	0	0	0	0	d = THREE
0	0	0	0	0	e = LOCAL_QUORUM
0	0	0	0	0	f = EACH_QUORUM
117	1073	0	2782	553	g = QUORUM
114	1029	0	2771		h = ALL

4.6 Overhead of OptCon

The overhead of the OptCon framework is plotted in the Figure 5, which gives the total delay, in milliseconds, for the operations performed by the OptCon wrapper for predicting the consistency level before making an actual call to the underlying data store. It is evident from the plot that the overhead from the various components of OptCon is negligible. Also this overhead does not affect our calculations of observed staleness, latency and other parameters, since these parameters are using isolated methods from the management bean interfaces.

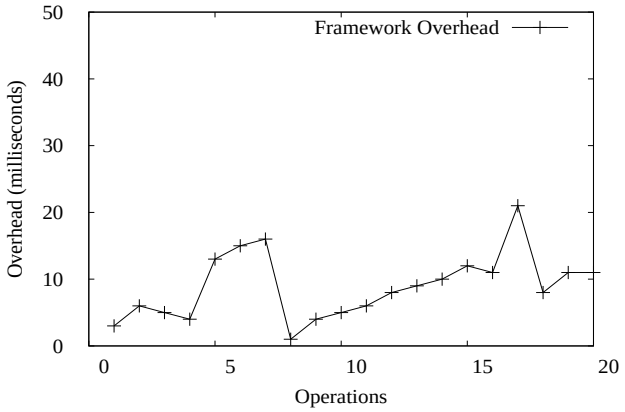

Figure 5: Framework Overhead

5 User Simulation Experiments

First we demonstrate the performance of OptCon with respect to variations in workload. We make sequential runs of various YCSB workloads - the core workloads, namely a, b, c, d, e, f, and Hot Spot Workloads (multiple transactional load) over a Cassandra cluster of seven large Amazon m3 instances, with replication factor of 3. We present a scatter plot (refer Figure 6) displaying the latency and staleness/observed consistency (represented by the average Γ score, i.e., the average of the per key Γ) under three subSLA rows - 1) SLA-1 with 20 ms threshold for latency and 5 ms for staleness, 2) SLA-2 with 50 ms threshold for latency and 2.5 ms for staleness, and 3) SLA-3 with 100 ms threshold for latency and 1 ms for staleness. It can be observed from the plot that OptCon outperforms fixed consistency settings in terms of satisfying the various subSLA thresholds. For example, under the fixed consistency level ONE READ/ANY WRITE the three subSLA's fail under all YCSB workloads. With READ/WRITE QUORUM, SLA-1 breaks under all possible workloads, while SLA-2 breaks under YCSB workload f and Hot Spot workloads. With ALL READ/QUORUM WRITE, SLA-1 and SLA-2 fails for all possible workloads, and SLA-3 fail under YCSB workloads d and e. On the other hand, OptCon satisfies the subSLA SLA-1 for all workload types, only breaking down under Hot Spot workloads, on the ground that the staleness exceeds the 5 ms threshold. Similarly, OptCon succeeds in satisfying SLA-2 for all YCSB workloads except for Hot Spots (latency at Hot Spot exceeds the 50 ms threshold). Also, the subSLA SLA-3 works for workload types a, b, c, e, f, and Hot Spot, but fails under workload d This shows that a particular subSLA may not be suitable for all possible workloads - the framework must work with a specific subSLA particularly designed according to the use case (defined here by the workload nature). This makes the case for having different subSLAs for different use cases, and a prediction framework (like ours) that can act upon the given subSLA and predict the matching consistency level for the workload type.

In order to study the adaptability of OptCon under various use cases, we have integrated it with the RUBBoS benchmark suite, developed under the JMOB project maintained by ObjectWeb. RUBBoS is a bulletin board web application modelled after SlashDot, an open source PERL-based online news forum application supporting discussion threads, story reviews and comments. RUBBoS has both PHP and Java Servlet implementations and the backend can be configured to support MongoDB, MYSQL, and Cassandra. We have integrated RUBBoS over Apache Cassandra 1.1.6 (latest Cassandra versions are not integrated with RUBBOS and OptCon doesnot have any dependency with the latest Cassandra versions)

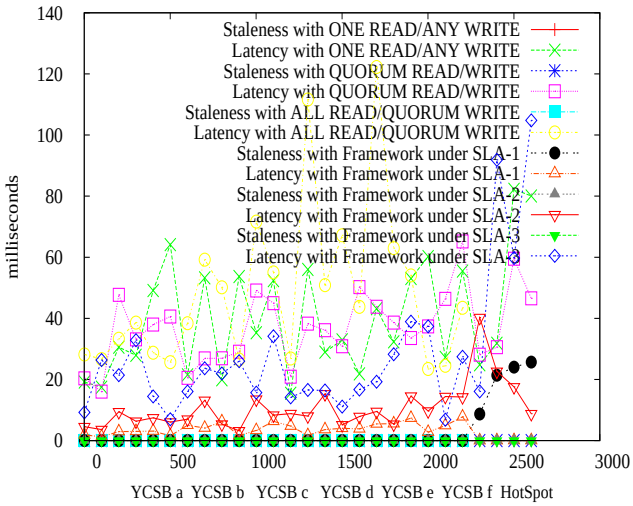

Figure 6: Performance With Variation of Workload under given subSLAs - a) SLA-1: latency 20ms, staleness 5ms, b) SLA-2: latency 50ms, staleness 2.5ms, and c) SLA-3: latency 100ms, staleness 1ms

with a replication factor of 5. Users can register to the website as normal users or authors, browse stories by categories, search by keywords, and submit new stories. According to surveys on browsing patterns by ObjectWeb, the majority of the users are expected to access the new stories and comments tables, which are accessible much more efficiently because of smaller size. RUBBoS has a daemon that periodically moves stories and comments from the new to the old tables contained in a 1.4 GB database. The application is used to generate the story and comment bodies with words from a given dictionary, and lengths ranging between 1KB and 8KB. The RUBBoS database is preloaded with data equivalent to 2 years worth of stories and comments. The framework has 500,000 total users stored in the backend, out of which 10 percent users have moderator access privilege.

Table 4: Chart Showing tradeoff in output parameters with Consistency Levels in User Study

subSLA-1		
Consistency	**Average,Mc Value**	**M Value**
ANY WRITE/ONE READ	2,4	68
READ/WRITE QUORUM	-0.7	41
Predicted Consistency	N/A	85
subSLA-2		
Consistency	**Average,Mc Value**	**M Value**
QUORUM,READ/ALL WRITE	0.5	70
Predicted Consistency	N/A	92
subSLA-3		
Consistency	**Average,Mc Value**	**M Value**
ALL,READ/QUORUM WRITE	-7.3	0
Predicted Consistency		100

We perform training on a dataset of 25600 rows gen-

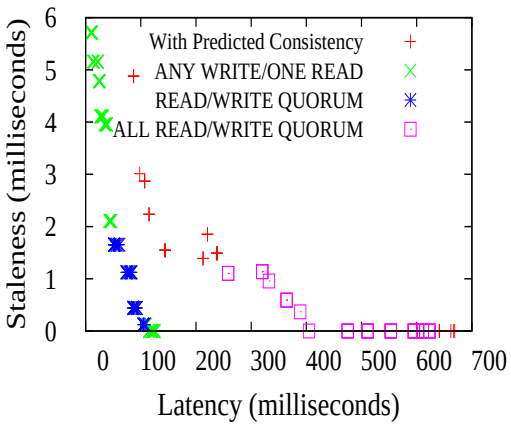

Figure 7: subSLA-1: Latency:250ms Staleness: 5ms

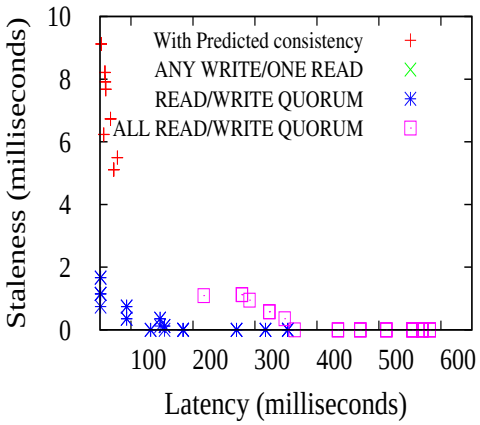

Figure 8: subSLA-2: Latency:100ms Staleness: 10ms

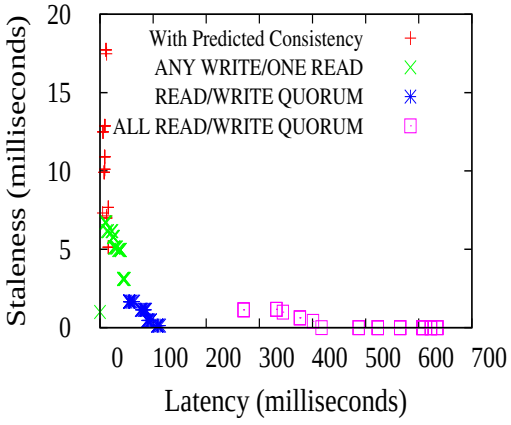

Figure 9: subSLA-3: Latency:20ms Staleness: 20ms

erated from simulations with various YCSB workloads as earlier (refer section 4.3). In the testing phase, we ran simulations with a series of standard user operations like User Registration, Login, Search, Browse, Submission, and Review using the representative servlet application, provided by RUBBoS on top of the Cassandra cluster. We preloaded the Cassandra database with a 16 GB data file, based on the data file obtained from the RUBBoS website, and performed a total of 200 operations comprising a set of standard operations representative of the standard combination of operations that a user performs on real world story board web applications, with consistency levels predicted by the Decision Tree based prediction framework.

The graphs in the Figures 7, 8, and 9 are two dimensional scatter plots, displaying the operations performed - each point represents the observed consistency/staleness values for the keys involved in consistency violations) against the latency measures for each operation under 3 sets of subSLAs. The blue dots represent read or write operations performed with consistency levels predicted by OptCon, whereas the red, grey, and yellow dots represent operations performed with fixed read/write consistency levels -namely
ANY WRITE/ONE READ, ALL READ/QUORUM WRITE, and QUORUM READ/ALL WRITE. Designating a particular Amazon instance as the NTP server and having the other instances synchronize with respect to that server, an observed clock skew of 1 ms is achieved in the cluster used for the experiments. The observed consistency level is measured in terms of the per-value (Γ)) score which, in turn, describes the maximum severity of any consistency violation associated with that value. We submit stories to the RUBBoS bulletin board for 4 operations, then browse the stories database, followed by submission of 2 more stories, and 2 reviews. Then sequentially, a browse operation was performed, followed by one story review, search for a story title, 2 more story reviews, 2 story submissions, and continue so on.

Figure 7 gives a comparison of performance measures under consistency levels predicted by OptCon against fixed consistency settings, given a subSLA specification of 250 ms latency bound and staleness bound of 5 ms. Figure 8 and 9 demonstrates performance of simulation experiments under subSLAs with latency bounds 100 and 20, and staleness bound of 10 and 20 ms respectively. OptCon fails to satisfy the strong consistency requirement, implied by the low staleness bound, and the high latency bound, given in the subSLA of Figure 7, in cases where the system applies fixed eventual consistency settings (i.e, ANY WRITE/ONE READ), resulting in the staleness exceeding the subSLA limit of ms. On the other hand, OptCon effectively succeeds in satisfying the subSLA bounds, when applying predicted stronger client level consistency settings, thus achieving low staleness values, while compromising on the latency front. Figure 9, with higher staleness and low latency

bound mandated by the subSLA, represents situations where the use case demands weak consistency settings. In this case, fixed strong consistency levels (i.e, ALL READ/QUORUM WRITE, and QUORUM READ/ALL WRITE) produce latency values way above the acceptable subSLA threshold of 20 ms. OptCon, however, achieves the latency subSLA by applying weaker consistency levels (eventual, i.e., ANY). Similarly, the figure 8 corresponds to subSLA specifications with comparatively weaker consistency setting and moderate latency limits, which fixed consistency settings fail to achieve in most cases, whereas OptCon satisfies the subSLA specifications in almost all cases.

We simulated heavy network congestion scenarios using the traffic shaping feature of Traffic Control, a linux-based tool for configuring network traffic, that controls the transmission rate by lowering the network bandwidth. We particularly used qdisc to enqueue incoming packets in pfifo (a pure First In, First Out queue) to store traffic when the network interface cannot handle the load momentarily. We also use netem, a network emulation tool to add fixed amount of delay to all packets going out of the local Ethernet. We applied network delays ranging from 500 to 100000 ms for some operations - resulting in high latency values, even with weak consistency setting, representing real world applications that work under heavy load and high network traffic. ln these cases, OptCon fails under our original subSLA specifications (i.e., latency bound of 101 ms) - the latency threshold has to be set appropriately high in such conditions of heavy network congestion.

The effectiveness of OptCon can be evaluated by two sets of metrics: 1) M, which measures the adaptability [45] [31] of the system under subSLA bounds, and 2) M_c, which measures the performance of the system with respect to the maximization objective (inspired by the average utility metric used by [40]), as a percentage improvement over fixed consistency level. M gives the percentage of cases which did not violate the subSLA bounds, given in the Table 4. The M values for OptCon under all the given subSLAs, representing the the blue dots in the Figures 7, 8, and 9, exceed the M values corresponding to the fixed consistency levels, given by the red, yellow, and grey points, for all the given subSLAs. From that it is evident that the approach of using dynamically predicted consistency levels obtained from OptCon gives a marked improvement over the fixed consistency levels approach. For the calculation of M_c, we compute the values $diff = Obs_{fixed} - Obs_{pred}$, where Obs_{pred} and Obs_{fixed} are the observed consistency (i.e, staleness metric values) obtained with predicted consistency levels C_{obs}, and with fixed consistency levels C_{fixed} (corresponding to the red, grey, and yellow dots respectively, for the same operation. Finally, M_c value, for a particular fixed consistency level C_{fixed}, is given as the average of all $diff$ values corresponding to C_{fixed}, i.e. $M_c = \sum_{diff}/N$. The M_c values corresponding to the figures 7, 8, and 9 is also given in the Table 4, except for the first subSLA, OptCon beats the fixed consistency approach, in terms of observed consistency values (decreased staleness, as evident from the positive M_c values). The combination of M and M_c values demonstrate that OptCon is at least as effective as an optimal combination of fixed consistency settings. On top of that, OptCon succeeds in producing effective matching consistency choices in cases where fixed consistency levels fail to satisfy the subSLA demands. The experiments are representative of the most common set of operations that users carry out on any desktop or web based applications. RUBBoS provides the underlying framework to simulate concurrent access, and interleaving user operations - thus the results we obtain can be used as a measure of how OptCon would fare, when integrated with any real world application. Table 4 displays how OptCon can be used effectively to make applications work under different latency-consistency combinations. Real time systems would operate mostly under the low latency-weak consistency band (refer Figure 9), whereas systems demanding stronger consistency level, such as transaction and batch processing applications, would be working predominantly in the high latency-strong consistency region (refer Figure 7).

6 Related Work

Eventual consistency was first used in Bayou [41], and later incorporated into a number of storage systems that implement quorum-based replication schemes: Amazon's Dynamo [12, 43], Cassandra [22], Voldemort [39] and Riak [1]. Dynamo, Cassandra and Riak can be configured to provide either eventual consistency or strong consistency using client-side consistency settings. In practice eventual consistency is preferred over strong consistency in scenarios where the system must maintain availability during network partitions, or when the application is latency-sensitive and able to tolerate occasional inconsistencies [3, 9].

Wada et al. [44] and Bermbach et al. [7] analyze consistency in commercial cloud storage systems and answered the question "how soon is eventual?" from a system-centric perspective that focuses on the convergence time of the replication protocol. Bailis et al. [6] and Rahman et al. [33] instead consider a client-centric perspective in which a consistency anomaly occurs only if differences in state among replicas lead to a client application actually observing a stale read. Staleness can be quantified by defining relaxed consistency properties that generalize Lamport's atomicity property for read/write

registers [24]. Aiyer et al. define k-atomicity, where k is a bound on version-based staleness (i.e., every read returns one of the last k updated values) [4]. Golab et al. define Δ-atomicity, where Δ is a bound on time-based staleness (i.e., every read returns a value at most Δ time units stale) [17]. The Γ consistency metric, which we use to define the staleness component of an SLA, is a remedy to the problem that Δ is undefined in scenarios where a value appears to be read before it is written due to clock skew [18]. Empirical measurements of Γ and Δ obtained using Cassandra are presented in [18, 33]. Bailis et al. present a probabilistic framework for predicting consistency in quorum-based key-value storage systems [6]. Their model uses simplified definitions of time and version-based staleness based upon the assumption that writes are not executed concurrently with other operations.

A large body of research deals with the problem of supporting various forms of stronger-than-eventual consistency in scalable storage systems and databases. The state machine replication paradigm achieves the strongest possible form of consistency by physically serializing operations [23]. Lamport's Paxos protocol is a quorum-based fault-tolerant protocol for state machine replication [25]. Mencius improves upon Paxos by ensuring better scalability and higher throughput under high client load using a rotating coordinator scheme [29]. A number of scalable fault-tolerant storage systems have been constructed using variations on Paxos: Spinnaker [35], Gaios [8], MDCC [20], and Spanner [11]. These systems use multiple instance of Paxos for scalability, and as a result they incorporate additional mechanisms, such as two-phase commit in Spanner, to support distributed transactions. Several other papers discuss techniques for supporting transactional semantics at scale, including reducing latency under low contention using "fast" Paxos [20], executing transactions using a combination of eventually consistent and strongly consistent operations [26, 27], providing eventually consistent transactions [10], and using static analysis to prevent conflicts among transactions [48]. Whereas Paxos-based and transactional systems aim to provide strong forms of consistency, our framework is intended to simplify the tuning of systems that support simple read and write operations with weak consistency, for example using sloppy quorums.

Relatively few systems provide mechanisms for fine-grained control over consistency. Yu and Vahdat propose a middleware layer for tunable availability and consistency tradeoffs (TACT) that uses three metrics to express consistency requirements with respect to read and write operations: numerical error, order error, and staleness [47]. Terry et al. present Pileus, a transactional key-value store that supports user-specified SLAs specified as a sequence of consistency and latency targets (sub-SLAs) ordered by a user-defined utility metric [40]. In contrast to these systems our framework can be applied on top of an existing eventually consistent data store.

7 Conclusion

OptCon automates the task of selecting the client-side consistency level in a distributed storage system, which is otherwise specified manually by skilled users or system administrators. This approach enables an intelligent combination of strong consistency, in certain feasible scenarios, and weaker forms of consistency in other cases, depending on the state of the network and storage system.

We experimented with three implementations of the Learning module of OptCon: a simple linear regression model, a decision tree classifier, and a Bayes network. In terms of accuracy the decision tree performed best, with an area under the ROC curve of 0.9857 and Gini coefficient of 0.9714. The Bayes network was the runner-up with an area under the ROC curve of around 0.9 and a Gini coefficient of 0.8. The linear regression model was a partial success in terms of ANOVA analysis, with p-values below 0.01 showing that latency is strongly dependent on the input variables, especially the read/write ratio. For staleness we observed a strong dependence on the read/write ratio only, and for the retransmission rate we observed no dependence. Thus, latency and staleness are most amenable to prediction using linear regression.

We compared the performance of OptCon against various fixed client-side consistency levels using the RUBBoS benchmark suite. The experimental results show that OptCon is able to predict a client-side consistency level that satisfies representative SLAs 85% of the time or better by providing either low latency or low staleness as required. In contrast, a fixed client-side consistency level can optimize for either latency or staleness, but not both.

Currently, to provide the users with minimum possible staleness, we choose the strongest consistency setting in case of multiple consistency options satisfying a given subSLA. We plan to use boosting [14] in such cases to obtain the matching consistency. In future plan to work on devising a mechanism for predicting fallback consistency levels in case of failure of the predicted consistency settings under partition.

References

[1] Basho riak. `http://basho.com/riak/`.

[2] M. A. The NoSQL ecosystem. In *In The Architecture of Open Source Applications*, page 185205, 2011.

[3] D. Abadi. Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. *Computer*, 45(2):37–42, Feb. 2012.

[4] A. Aiyer, L. Alvisi, and R. A. Bazzi. On the availability of non-strict quorum systems. In *Proc. International Symposium on Distributed Computing (DISC)*, pages 48–62, 2005.

[5] M. S. Ardekani and D. B. Terry. A self-configurable geo-replicated cloud storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 367–381, Broomfield, CO, Oct. 2014. USENIX Association.

[6] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endow.*, 5(8):776–787, Apr. 2012.

[7] D. Bermbach and S. Tai. Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior. In *Proc. Workshop on Middleware for Service Oriented Computing (MW4SOC)*, 2011.

[8] W. J. Bolosky, D. Bradshaw, R. B. Haagens, N. P. Kusters, and P. Li. Paxos replicated state machines as the basis of a high-performance data store. In *Proc. of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 11–11, Berkeley, CA, USA, 2011. USENIX Association.

[9] E. A. Brewer. Towards robust distributed systems (Invited Talk). In *Proc. of the 19th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2000.

[10] S. Burckhardt, D. Leijen, M. Fähndrich, and M. Sagiv. Eventually consistent transactions. In *Proc. of the 21st European Conference on Programming Languages and Systems*, ESOP'12, pages 67–86, Berlin, Heidelberg, 2012. Springer-Verlag.

[11] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *Proc. of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 251–264, Berkeley, CA, USA, 2012. USENIX Association.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, Oct. 2007.

[13] T. Everts. Web performance today.

[14] P. Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, New York, NY, USA, 2012.

[15] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, SOSP '79, pages 150–162, New York, NY, USA, 1979. ACM.

[16] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[17] W. Golab, X. Li, and M. A. Shah. Analyzing consistency properties for fun and profit. In *Proc. of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, pages 197–206, New York, NY, USA, 2011. ACM.

[18] W. M. Golab, M. R. Rahman, A. AuYoung, K. Keeton, J. J. Wylie, and I. Gupta. Client-centric benchmarking of eventual consistency for cloud storage systems. In *ICDCS*, page 28, 2014.

[19] C. Hale and R. Kennedy. Using riak at yammer.

[20] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete. MDCC: multi-data center consistency. In *Proc. of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 113–126, New York, NY, USA, 2013. ACM.

[21] K. LaCurts, J. C. Mogul, H. Balakrishnan, and Y. Turner. Cicada: Introducing Predictive Guarantees for Cloud Networks. In *HotCloud*, ,, June 2014.

[22] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.

[23] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558, 1978.

[24] L. Lamport. On interprocess communication. *Distributed Computing*, 1(2):77–85, 1986.

[25] L. Lamport. Paxos made simple, fast, and byzantine. In *OPODIS*, pages 7–9, 2002.

[26] C. Li, J. Leitão, A. Clement, N. Preguiça, R. Rodrigues, and V. Vafeiadis. Automating the choice of consistency levels in replicated systems. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 281–292, Philadelphia, PA, June 2014. USENIX Association.

[27] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *Proc. of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 265–278, Berkeley, CA, USA, 2012. USENIX Association.

[28] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[29] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: Building efficient replicated state machines for WANs. In *Proc. of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 369–384, Berkeley, CA, USA, 2008. USENIX Association.

[30] C. Meiklejohn. Riak PG: Distributed process groups on dynamo-style distributed storage. In *Proc. of the Twelfth ACM SIGPLAN Workshop on Erlang*, Erlang '13, pages 27–32, New York, NY, USA, 2013. ACM.

[31] A. Paschke and E. Schnappinger-Gerull. A categorization scheme for SLA metrics. *Service Oriented Electronic Commerce*, 80:25–40, 2006.

[32] E. Plugge, T. Hawkins, and P. Membrey. *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*. Apress, Berkely, CA, USA, 1st edition, 2010.

[33] M. R. Rahman, W. , A. AuYoung, K. Keeton, and J. J. Wylie. Toward a principled framework for benchmarking consistency. In *Proc. of the Eighth USENIX Conference on Hot Topics in System Dependability*, HotDep'12, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.

[34] M. R. Rahman, L. Tseng, S. Nguyen, I. Gupta, and N. Vaidya. Probabilistic CAP and timely adaptive key-value stores. 2014.

[35] J. Rao, E. J. Shekita, and S. Tata. Using paxos to build a scalable, consistent, and highly available datastore. *PVLDB*, 4(4):243, 2011.

[36] L. Ravindranath, J. Padhye, R. Mahajan, and H. Balakrishnan. Timecard: Controlling user-perceived delays in server-based mobile applications. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 85–100, New York, NY, USA, 2013. ACM.

[37] T. Schütt, F. Schintke, and A. Reinefeld. Scalaris: Reliable transactional P2P key/value store. In *Proceedings of the 7th ACM SIGPLAN Workshop on ERLANG*, ERLANG '08, pages 41–48, New York, NY, USA, 2008. ACM.

[38] M. Stonebraker. Urban myths about sql, June 2010.

[39] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah. Serving large-scale batch computed data with project Voldemort. In *Proc. of the 10th USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[40] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *Proc. of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 309–324, New York, NY, USA, 2013. ACM.

[41] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, pages 172–182, 1995.

[42] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. *SIGCOMM Comput. Commun. Rev.*, 39(4):303–314, Aug. 2009.

[43] W. Vogels. Eventually consistent. *ACM Queue*, 6(6):14–19, 2008.

[44] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu. Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In *Proc. Conference on Innovative Data Systems Research (CIDR)*, pages 134–143, 2011.

[45] Wikipedia. High availability — wikipedia, the free encyclopedia, 2014. [Online; accessed 24-October-2014].

[46] K. Winstein and H. Balakrishnan. Tcp ex machina: Computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 123–134, New York, NY, USA, 2013. ACM.

[47] H. Yu and A. Vahdat. Building replicated internet services using TACT: A toolkit for tunable availability and consistency tradeoffs. In *WECWIS*, pages 75–84, 2000.

[48] Y. Zhang, R. Power, S. Zhou, Y. Sovran, M. K. Aguilera, and J. Li. Transaction chains: Achieving serializability with low latency in geo-distributed storage systems. In *Proc. of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 276–291, New York, NY, USA, 2013. ACM.