YCSB-D: Benchmarking Adaptive Frameworks With Dynamic Workload Variations

Subhajit Sidhanta
Louisiana State University
ssidha1@lsu.edu

Supratik Mukhopadhyay
Louisiana State University
supratik@csc.lsu.edu

Wojciech Golab
University of Waterloo
wgolab@uwaterloo.ca

ABSTRACT

We demonstrate YCSB-D, a tool that builds upon YCSB (Yahoo Cloud Serving Benchmark suite) to assist users in simulating dynamic variations in workloads. YCSB-D allows users to run a sequence of YCSB workloads on a distributed datastore over long periods of time, without requiring users to manually change the workload configuration in individual nodes each time the workload type needs to be modified. YCSB-D can be used to simulate workloads in real world systems that use distributed datastores (like Netflix, Youtube, Facebook, Amazon, BitTorrent, etc.), which reportedly exhibit different workload characteristics during different times of the day. Distributed datastores, operating under SLAs (Service Level Agreements), must adapt their configurations to such workload variations on a per-operation basis, to avoid SLA violations, in terms of the observed latency and staleness (i.e., how old the result is). A considerable body of database and systems research aims at developing automated adaptive frameworks, that can tune distributed datastores to dynamic workload variations. The dynamic workload variations simulated with YCSB-D can be used to evaluate the adaptability of such frameworks to changing workload characteristics. We demonstrate the ability of YCSB-D to evaluate the adaptability of OptCon, an automated framework, that tunes the consistency settings of Cassandra with respect to the latency and staleness thresholds in an SLA.

1. INTRODUCTION

YCSB [3] is widely used to simulate standard benchmark workloads for cloud-based systems, like Cassandra, MongoDB, Hbase, etc. [5, 2]. The YCSB suite comprises a workload generator, that can simulate a set of core workload types, characterized by parameters, like read-write proportions, request distribution, target throughput, etc., emulating different types of real world workload scenarios. To simulate scenarios where concurrent applications access the datastore from multiple nodes, YCSB can be set up to run as a collection of parallel client processes, where each node in a cluster runs a separate YCSB process. For executing YCSB in parallel mode, a user must manually start the YCSB workload executor from the command line at each individual node of the cluster. Workloads in real world web-based applications (like Netflix, Youtube, Facebook, Amazon, BitTorrent, etc.), that use distributed datastores, widely vary over time [9]. Workload variation for such systems typically exhibit a specific pattern; the workload type varies in a characteristic fashion over time. For example, Netflix [9] observes that the network traffic for its applications reaches almost 37% of Internet traffic during peak workload hours. For benchmarking such systems, a user must execute benchmark workloads for the target system in an uninterrupted back-to-back *sequence*, following the characteristic workload variation pattern specific to the use case.

We address the problem of allowing users to specify a back-to-back sequence of YCSB workloads, and execute the sequence in parallel in an uninterrupted manner. This involves the following sequence of steps to be performed at each individual node: 1) a given core workload is started from the YCSB command line, 2) after a specific time period the above workload must be stopped, 3) the next workload must be started, without any interruption (transition delay), from the command line, 3) iterate over step 1 to 2 uninterrupted, until the workload execution reaches the end of the benchmark period. The transition in step 2 must occur smoothly, without any interruption (delay), according to the characteristic variation pattern [9]. To execute a sequence of parallel workloads back-to-back, the YCSB process running in each individual node of a cluster needs to be stopped at the same time, and then simultaneously restarted at each node.

Performing the above sequence of steps at multiple nodes in a cluster at the same time, using the command line YCSB client can be difficult, even with YCSB++ [6]. For parallel YCSB processes executing on a cluster, YCSB++ uses ZooKeeper-based barrier synchronization to synchronize the client processes in individual nodes of the cluster. But YCSB++ does not support dynamically changing workloads. To run a different workload, the YCSB++ process at each node has to be manually stopped and restarted using the command line tool, which can result in transition delays. But the above transition from step 2 to step 3 must be instantaneous, to simulate smooth transition from one workload to another. Hence, benchmark statistics generated using the above technique may not accurately reflect that in real world workload scenarios, which are typically characterized by smooth transition from one characteristic workload to another [9]. We propose YCSB-D, a tool that allows users to automatically simulate a smooth variation pattern for parallel workloads executing over a target distributed datastore, running on a cluster of nodes. The tool allows users to specify a required pattern of variation in the workload over a given time period through an additional configuration parameter to the YCSB client. In addition, the tool can be used to simulate real world scenarios, where the workload transitions gradually over time according to a characteristic pattern.

The control flow for YCSB is as follows: 1) the YCSB client module accepts workload configuration parameters, like number of threads, selected core workload, etc., from the command line, 2) the client module configures the selected core workload using the above parameters, 3) the client module instantiates the specified number of threads to execute the selected core workload, 4) the control is then passed to the workload executor which executes the threads. Since the selected core workload is configured and initialized by the client module before the control passes to the workload executor, only a fixed core workload can be executed at a time by YCSB. YCSB-D integrates steps 2 and 3 with step 4, allowing the YCSB executor module to configure, initialize, and execute a sequence of core workloads dynamically. The above goal can not be achieved by: 1) tweaking the YCSB configuration files to specify the workload sequence, or by 2) developing a script, an interface layer, or a wrapper, to execute the YCSB command line tool back-to-back with a different workload configuration each time. For executing a sequence of YCSB workloads in parallel, the latter technique will require the users to use the script/interface layer/wrapper to restart YCSB processes at each individual node in a cluster, that can result in transition delays in the execution of the workload sequence.

Contribution: We demonstrate YCSB-D, a tool that can simulate *dynamic* variations in YCSB workloads, and display the resulting variations in observed latency and throughput over time as a time series graph. YCSB-D provides a web-based user interface that comprises an input form that allows users to define a sequence of YCSB workloads to be executed over a given time period. The user can divide the total duration of YCSB execution into a sequence of smaller time periods, and configure different YCSB workloads to be executed in each time period. The workload sequence is captured from the input form as a sequence of tuples; each tuple specifies a selected core YCSB workload type (a, b, c, etc.,) and its duration of execution. Users can pass input parameters to YCSB-D, like number of threads, record count, request distribution, etc., along with a workload sequence. YCSB-D only modifies the client module that configures the workloads to be executed, keeping the underlying architecture unchanged. Hence, YCSB-D can work with any target system for which YCSB provides a connector (like Cassandra, Hbase, MongoDB, etc).

A considerable body of database and systems research [11, 10, 1] aims at developing automated adaptive frameworks, that can tune distributed datastores to adapt to workload variations, under performance thresholds specified in the SLAs. Such frameworks can automatically tune the consistency settings of an underlying datastore, thus helping the client applications satisfy the SLA under dynamic workload variations. Testing the adaptability of such frameworks requires benchmarking them with an exhaustive set of charac-

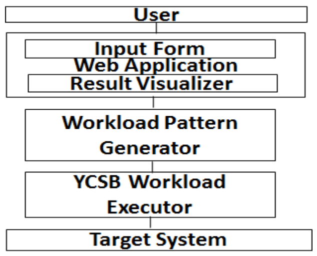
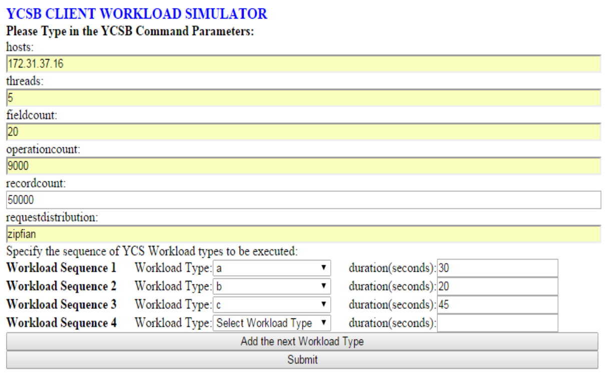


Figure 1: The architecture of YCSB-D


YCSB CLIENT WORKLOAD SIMULATOR

Please Type in the YCSB Command Parameters:

hosts: 172.31.37.16

threads: 5

fieldcount: 20

operationcount: 9000

recordcount: 50000

requestdistribution: zipfian

Specify the sequence of YCS Workload types to be executed:

Workload Sequence 1	Workload Type: a	duration(seconds): 30
Workload Sequence 2	Workload Type: b	duration(seconds): 20
Workload Sequence 3	Workload Type: c	duration(seconds): 45
Workload Sequence 4	Workload Type: Select Workload Type	duration(seconds):

Figure 2: YCSB-D User Interface

teristic workload variations, simulated with YCSB, which is the standard benchmarking tool for distributed datastores. We propose leveraging the capability of YCSB-D to simulate possible variation patterns of YCSB workloads to evaluate the ability of an adaptive framework to adjust to dynamic workload variations, avoiding violation of the performance thresholds specified in the SLA. Thus, YCSB-D can effectively benchmark adaptive tuning frameworks [11, 10, 1] against varying workload characteristics, under a given SLA. Following [10], we provide a metric M to benchmark the adaptability of a target system to dynamic workload variations; M is given as the percentage of the operations which did not result in SLA violations. For this demonstration, YCSB-D is integrated with OptCon [7], an automated adaptive framework that can tune the consistency settings of a distributed datastore, such that the observed latency and staleness match the tolerance thresholds in the SLA, under a given workload and network condition. Among the adaptive tuning frameworks [11, 10, 1], OptCon is the only open-source framework that we could find. We use the dynamic workload variations simulated with YCSB-D to benchmark OptCon's adaptability to varying workload, under the latency and staleness thresholds given in the SLA.

2. DESIGN AND IMPLEMENTATION

Design: Figure 1 presents the architecture of YCSB-D. YCSB-D modifies the code for the YCSB client, which involves complete restructuring of the execution flow of YCSB, allowing a sequence of workloads to be executed over predefined time periods. YCSB-D comprises a client module that accepts input parameters from users, like number of threads, operation count, etc. The client module passes the input parameters to the Workload Generator module, which generates a sequence of YCSB workloads to be executed. The YCSB Workload Executor module executes the above workload sequence over a target system. The Result Visualizer module displays the variation in the observed latency and throughput over the duration of the execution of the workload sequence.

Implementation: YCSB-D implementation comprises two artifacts. 1) A Java-based web application that comprises an input form that collects the input parameters from the users, and calls the YCSB-D client. 2) A modified version of the YCSB framework that alters the source code of the YCSB client to accept a workload variation pattern. 3) A Workload Generator, that defines a sequence of core workloads to be executed over a given time period, and 4) a Workload Executor module that allows execution of a sequence of core workloads. Figure 2 presents the web interface for YCSB-D. Figure 2 shows the input form, that the client module uses to submit the input parameters. Using the input form, the user can easily define workload variation patterns as follows. Users can choose a core workload type w from a dropdown menu, and specify the duration of execution of w in the adjacent text box. The dropdown menu allows users to select a workload from the choice of core workloads available in the YCSB suite. He can then add another workload type to the workload sequence, by clicking on the $\langle Add\ the\ Next\ Workload \rangle$ button, and specifying the workload type and duration as above. The workload sequence is captured by the web application as a sequence c of tuples; each tuple $\langle w, d \rangle$ comprises a character w that represents the selected workload type, and a floating point parameter d that denotes the duration for the execution of w. The above sequence c is passed to the workload generator module, which then generates a sequence of core YCSB workloads to be executed.

Result Visualizer The Result Visualizer module comprises a web page that displays the observed latency and throughput for the benchmark operations in the form of a time series. From the viewpoint of the user of a distributed datastore, important performance parameters are latency and client-observed *consistency anomalies* [10], i.e., anomalies in the result of an operation observed by a client application. Consistency anomalies are measured in terms of client-centric *staleness* [10], i.e., how stale (old) is the version of the data item (observed by a client application) with respect to the most recent version. The Result Visualizer also displays the observed consistency, i.e., staleness, computed in terms of the $\mathbf{\Gamma}$ metric of Golab et. al. [4]. As demonstrated in [4], $\mathbf{\Gamma}$ is preferred over other client-centric staleness measures for its proven sensitivity to workload parameters.

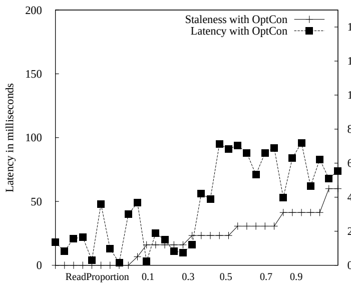
3. DEMONSTRATION

Benchmarking Adaptive Tuning Frameworks The performance of cloud-based applications and services, that use distributed datastores, is affected by the consistency settings applied to the datastore. According to the consistency setting applied, a distributed datastore waits for coordination among a specific number of replicas containing copies (i.e., versions) of the data item accessed by a given operation [8]. With a weak consistency setting for an operation, the datastore waits for coordination among a smaller number of replicas; hence there is a higher chance of yielding a stale result (since the most recent update may not have reached all the replicas through replica synchronization techniques). On the other hand, stronger consistency settings yield less stale results (more recent version), since they enforce coordination among a larger number of replicas. The latency for a given operation depends on the waiting time for the above coordination; hence, in turn, depends on the consistency setting applied. Thus, both staleness and latency for an operation vary with the choice of consistency setting. The chosen consistency setting must be *matching* with respect to the latency and staleness thresholds specified in the given SLA, i.e., it must be weak enough to satisfy the latency threshold, while being strong enough to satisfy the staleness threshold. A typical use case at Netflix [9] requires real-time response; hence the SLA typically comprises a low tolerance threshold for latency and a higher threshold for staleness. For the given SLA, under a peak-hour (prime-time) workload, the matching choice is a weak consistency setting. If the developer applies stronger consistency settings, the resulting high latency might violate the SLA.

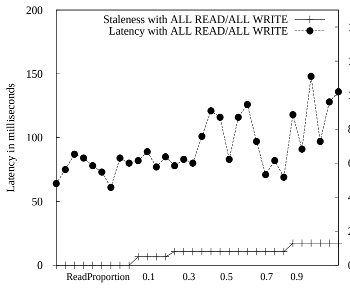
To automate the process of consistency tuning in distributed datastores, a number of prototype adaptive tuning frameworks have been developed, like TACT [11], Pileus [10], Tuba [1], OptCon [7], etc. Such frameworks can adapt the underlying datastore to variations in the workload, thus ensuring that the system always satisfies the SLA. However, testing such capabilities require the framework to be subjected to an exhaustive set of possible variations in the workload. We propose that the adaptability of such adaptive frameworks can be evaluated with dynamic variations in YCSB benchmark workload, simulated using the YCSB-D tool. YCSB-D can be used to simulate real world scenarios, characterized by specific patterns of workload variations. Thus, YCSB-D can be used to test if an adaptive framework can adjust the underlying datastore to satisfy the SLA under varying workloads; hence, YCSB-D can be used to benchmark the above mentioned adaptive frameworks. Following [10], we quantify the adaptability of adaptive frameworks by the metric M, which computes the percentage of operations which did not violate the SLA.

Demonstration Setup: OptCon [7] is an open-source example of an automated adaptive framework, that tunes Cassandra with respect to the latency and staleness thresholds in the SLA. We use YCSB-D to evaluate the adaptability of OptCon to dynamically varying YCSB workloads. The results obtained by running dynamically varying workloads (simulated with YCSB-D) on OptCon can be quantified in terms of the M-metric, that measures the adaptability of the framework (i.e., OptCon). YCSB-D is built as an extension of YCSB, which is a widely accepted benchmark for cloud-based systems and distributed datastores. Hence, YCSB-D can be used to evaluate adaptive frameworks that tune distributed datastores. However, OptCon is the only open-source available adaptive framework that we could find. The source code for OptCon can be found at: `https://github.com/ssidhanta/HectorCient/`. We have run YCSB-D on a testbed of 20 Amazon ec2 small instances, located in the same Ec2 region, running Ubuntu 13.10, loaded with Cassandra 2.1.2 with a replication factor of 3 (the number of replicas per data item). The demonstration can be accessed using the url: `http://ec2-52-36-221-215.us-west-2.compute.amazonaws.com:8080/YCSBDemo`

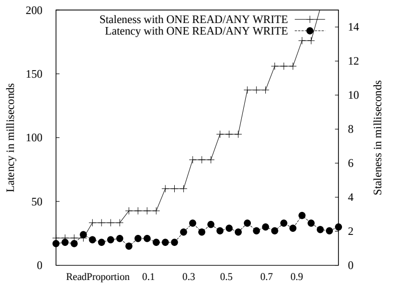
Demonstrating Adaptability of OptCon: We demonstrate that YCSB-D can effectively evaluate the adaptability



(a) With OptCon; $M = 100$



(b) With Fixed Strong Consistency Settings; $M = 70$



(c) With Fixed Weak Consistency Settings; $M = 45$

Figure 3: Adaptability of OptCon to Varying Workload (Read Proportion) under the SLA (Latency:250ms Staleness: 5ms)

of OptCon to varying YCSB workloads. YCSB workloads are executed on top of a Cassandra cluster, varying the core workload type using YCSB-D. The staleness and latency thresholds for a given SLA are supplied to OptCon in a configuration file. In Figures 3(a), 3(b), and 3(c) we plot the read proportion for the workload along the x axis, the observed latency (obtained from the Result Visualizer) along the primary y axis, and the observed staleness (the $\mathbf{\Gamma}$ values obtained from the Result Visualizer) along the secondary y axis. In Figure 3(a), OptCon satisfies the SLA thresholds in 100% cases; hence, $M = 100$, whereas with manually chosen fixed consistency settings in Figures 3(b) and 3(c), M is 70 and 45, respectively. This demonstrates that OptCon can adapt to varying workloads, tuning the consistency setting of the target datastore (Cassandra) according to the current workload.

For a specific read proportion, only a subset of all possible consistency settings (ranging from strong to weak) is *matching*, i.e., satisfies latency and staleness thresholds in a given SLA. Under a given SLA, OptCon tunes the datastore with consistency settings chosen from the matching set of consistency settings for each read proportion (refer Figure 3(a)). As shown by the results in [4], the frequency of stale results (i.e., higher Γ scores) increases with increasing read proportion in the workload. Hence with higher read proportion, $RW > 0.5$ in in the right-half of the x axis (refer to Figure 3(a)), stronger read consistency settings are required to return less stale results. Hence fixed manually chosen strong consistency setting satisfies the SLA in the right-half of Figure 3(b), whereas fixed weak consistency settings result in violation of the SLA in the right-half of Figure 3(c). OptCon adapts to higher read proportion by tuning the datastore to operate under strong consistency settings, and satisfies the SLA for all read proportions of the workload in Figure 3(a), i.e., $M = 100$. With lower read proportions in the left-half of the x axes, the frequency of stale read results are smaller [4]. In this case, weaker consistency settings are sufficient for returning consistent results. Hence, fixed weak consistency setting satisfies the SLA in the left-half of Figure 3(c), whereas fixed strong consistency setting results in violation of the SLA in the left-half of Figure 3(b). OptCon applies weak consistency settings to achieve the staleness and latency thresholds in the SLA for all workloads in Figure 3(a).

The latency and staleness plots in the Result Visualizer can be closely analyzed to observe whether OptCon can adjust to workload variations in a timely fashion, to eliminate the possibility of the users noticing SLA violations, in terms of the staleness and latency thresholds. Thus, YCSB-D can be used to obtain deeper insights on the adaptability of automated tuning frameworks against real time variations in workload. The database and systems community can benefit from YCSB-D, which can serve as a benchmarking tool for testing the ability of a framework to adapt a target datastore to variations in workload.

4. REFERENCES

[1] M. S. Ardekani and D. B. Terry. A self-configurable geo-replicated cloud storage system. In *OSDI 14*.

[2] K. Chodorow and M. Dirolf. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2010.

[3] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *SoCC '10*.

[4] 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.

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

[6] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi. Ycsb++: Benchmarking and performance debugging advanced features in scalable table stores. In *SOCC '11*.

[7] S. Sidhanta, W. Golab, S. Mukhopadhyay, and S. Basu. OptCon: An Adaptable SLA-Aware Consistency Tuning Framework for Quorum-based Stores. Technical report.

[8] S. Sivasubramanian. Amazon dynamodb: a seamlessly scalable non-relational database service. In K. S. Candan, Y. C. 0001, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *SIGMOD Conference*, pages 729–730. ACM, 2012.

[9] T. Sprangler. Netflix bandwidth usage climbs to nearly 37% of internet traffic at peak hours. `http://variety.com/2015/digital/news/netflix-bandwidth-usage-internet-traffic-1201507187/`, 2015.

[10] 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 *SOSP '13*.

[11] 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.