Hoagie: A Database and Workload Generator using Published Specifications

Shahram Ghandeharizadeh, Haoyu Huang
Database Laboratory Technical Report 2018-10
Computer Science Department, USC
Los Angeles, California 90089-0781
{shahram,haoyuhua}@usc.edu

Abstract

Hoagie is a plug-n-play workload and database generator to evaluate novel system architectures, design decisions, protocols, and algorithms. It uses published specifications to create a database of data items and a workload that references these data items. Hoagie's modular design enables an experimentalist to use it either offline or online. In offline mode, Hoagie outputs a trace file that can be used to issue requests to a target system. In online mode, Hoagie is plugged into an existing benchmark that invokes it to generate requests one at a time to its target system. We have made Hoagie open source to foster its future development.

1 Introduction

System designers, architects, and practitioners require realistic workloads to evaluate novel system design decisions including protocols and algorithms. A workload includes both a database of data items and a request pattern that references these data items. Companies are hesitant to release anonymized sample databases and/or traces of requests to foster research since they may leak sensitive information unintentionally. Studies have shown that an attacker may apply an inference technique to an anonymous trace to reveal real user identities. For example, AOL anonymized user identities and released a search data set in 2006. The release was intentional for research purposes. However, individual identities were unveiled by cross-referencing the data set with phone-book listings [20]. Subsequently, AOL was sued and required to pay $5000 to each person in the data set for violating the Electronic Communications Privacy Act. More recently, Facebook was fined over Cambridge Analytica leak [21] that may have impacted 87 million users.

This paper argues for a database and a workload generator that uses published distributions. Hoagie implements this using specifications provided by Facebook [4]. The motivation for Hoagie is several folds. First, companies are publishing specifications of their database and workloads instead of releasing databases and traces. For example, Facebook provides these specifications in several studies [3,4,9,16,17]. Second, the scarcity of traces has caused researchers to use only one trace to evaluate their design decisions, resulting in studies that are not convincing. For example, George et al. [1] report that the workload characteristics of scheduler traces released by Google in 2011 are outliers, affecting 450 publications that evaluate their novel techniques using Google traces alone. Third, popular benchmarks such as TPC-C may not be representative of today's transaction processing. TPC-C [22] was created in the early 1990s and its workload is write-heavy (92% of transactions are writes). As another example, the popular YCSB [10] benchmark database is a table of fix sized data items. Its simple workload consists of a mix of data item get, scan, insert, and update commands.

In this study, we propose a database and a workload generator named Hoagie [14]. Hoagie inputs key characteristics of published workloads to output (a) a realistic database of data items, and (b) a sequence of requests that reference these data items. Its input should be based on published statistical modeling of production workloads. One example is Facebook's published characteristics of its database and request access pattern [3, 4, 9, 16, 17]. We have used Hoagie in Gemini [13] to evaluate several crash recovery protocols for persistent caches.

The current implementation of Hoagie has several limitations. First, it is specific to Facebook's published specifications. Second, it does not scale out to use multiple nodes since clock skews may impact the correctness of modeled distributions, e.g., the inter-arrival time distribution. Third, the database size grows unbounded as Hoagie models an evolving working set. A solution for these limitations constitutes our future research, see Section 6.

The rest of this paper is organized as follows. Section 2 details published workloads. Section 3 describes challenges and solutions of implementing these in Hoagie. We present two use cases of Hoagie in Section 4. Section 4 provides related work and Section 6 presents our future research directions.

2 Workload Characteristics

Hoagie models workload characteristics based on publicly available specifications. Example characteristics include number of data items, size of a data item, popularity skew of references to data items, inter-arrival time between requests, etc. We describe these characteristics below.

2.1 Database

A database consists of a set of data items. A data item is uniquely identified by a key. Its size varies across different workloads. An OLTP workload's data item size may be measured in bytes while the data item size of a video streaming application may be in mega-bytes [15]. Facebook reports its key-size and value-size follow the generalized extreme value distribution and the generalized pareto distribution, respectively. Its mean key-size is 36 bytes and mean value-size is 329 bytes. Hoagie is designed to input distribution of data item sizes to produce a database that approximates the characteristics of a real database.

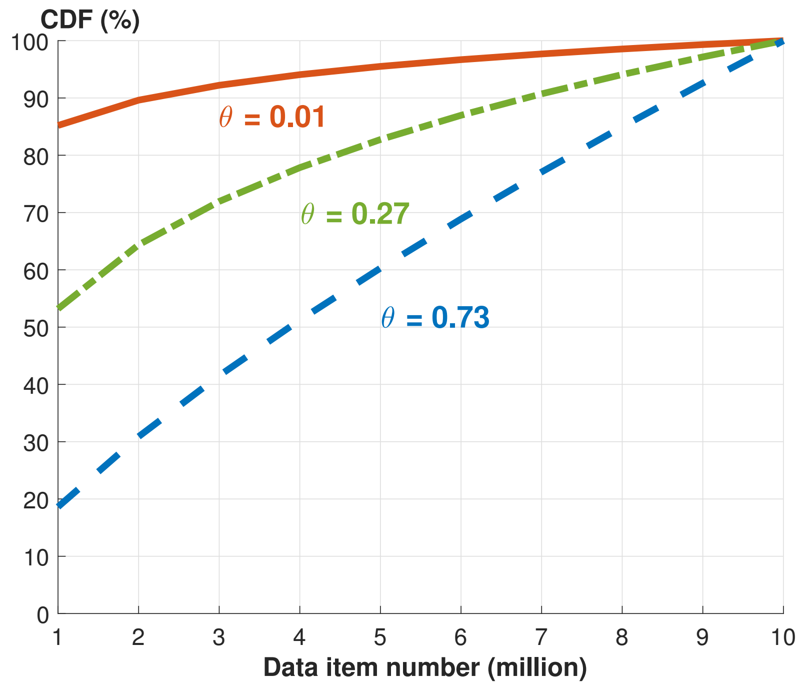

Figure 1: Popularity skew modeled by Zipfian Distribution with 10 million data items generated by Hoagie.

2.2 Popularity Skew

Real-world workloads [4, 10] observe a highly skewed access pattern. A large percentage of requests concentrate on a small portion of the data items. The popularity of a key decreases exponentially. Facebook reports that 90% of requests reference 10% of keys and a long tail where 30% of keys are referenced by less than 1% of requests.

Many benchmarks and workload generators (including Hoagie) use the Zipfian distribution to model the popularity skew. Zipfian distribution references a data item i drawn from a population of M data items numbered $[1, M]$ with probability[1]:

$$p_i(M, \theta) = \frac{1/i^{(1-\theta)}}{\sum_{m=1}^{M} 1/m^{(1-\theta)}}. \quad (1)$$

The parameter θ controls the skew. A smaller θ translates to a higher skew. With θ=0.01, 85% of requests reference 10% of keys, see Fig. 1. When a database is partitioned across multiple nodes, D-Zipfian [6] allows each node to generate requests independently on a disjoint set of data items while respecting the same Zipfian distribution.

2.3 Inter-arrival Time

The duration between two consecutive requests generated by Hoagie is inter-arrival time. A typical inter-arrival time distribution exhibits a diurnal pattern. A shorter inter-arrival time

[1]Other studies [8,10] use θ instead of $1-\theta$. For example, the distribution produced with 0.27 by equation one is identical to that produced using the alternative definition with 0.73.

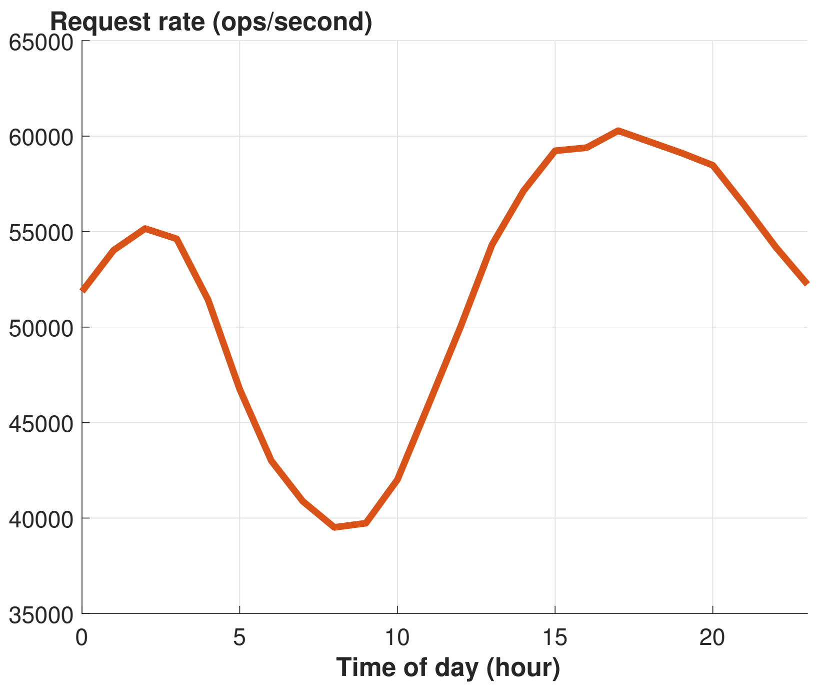

Figure 2: Request rate of Facebook workload [4].

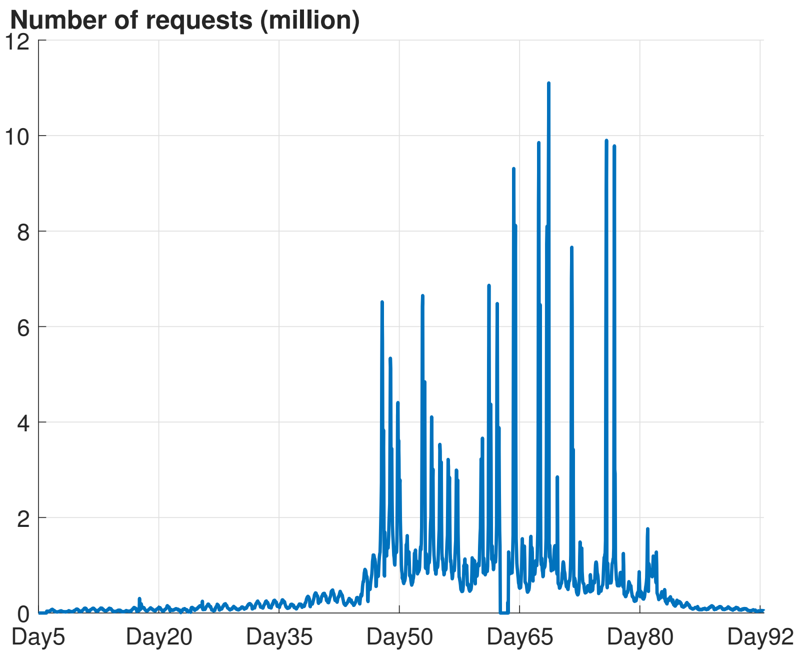

Figure 3: Request rate of 92 days during WorldCup '98 [2].

results in a higher request rate. For example, a data store may receive a higher request rate during the day and a lower request rate during the night. The inter-arrival time during peak hours can be orders of magnitude lower than off-peak hours. Fig. 2 shows the request rate per hour during a day drawn from Facebook workload [4]. Fig. 3 shows the request rate per hour during WorldCup '98 [2]. The peak load is 11 million requests per hour, which is an order of magnitude higher than the request rate during off-peak hours.

2.4 Working Set

A working set [11] is defined as a set of data items referenced by the application repeatedly. The working set size is determined by the request rate and the percentage of referenced unique data items at a time. Facebook reports a constant percentage of referenced unique data items irrespective of load changes, e.g., 20.7% of keys are unique for requests grouped by hour in a 5-day trace [4]. Moreover, its workload may exhibit either a temporal or spatial locality of reference:

2.4.1 Temporal locality

The temporal locality results in the popularity of a data item to change over time. A social networking site may observe a strong temporal locality that causes the popularity of a data item to decay exponentially over time. For example, a new post draws lots of attention once created and its number of views decreases drastically after a few hours. Facebook reports 88.5% of keys are reused within an hour, but only 4% within two hours [4].

2.4.2 Spatial Locality

A subset of a working set may be referenced together with a high probability, termed correlated references. For example, when a socialite looks up a friend's profile, then she may also look up the friend's latest photos. Due to the lack of published statistics, Hoagie does not model spatial locality at the time of this study.

3 Workload Generator

This section illustrates how Hoagie is tailored per Facebook's published specifications [4] to generate both a database of data items and requests that reference these data items. Hoagie's database contains a set of data items identified by strictly increasing integer values. Hoagie generates a trace on an hourly basis since Facebook provides per-hour statistics. A one-hour working set contains 54 million data items on average. Hoagie requires a user to specify the read to write ratio, a trace duration in hours, and a distribution for each of the workload characteristics. A data item is uniquely identified by a 64-bit integer key and associated with a key-size and a value-size in bytes. A request references a single data item and contains a timestamp indicating when the request should be issued. Below, we describe the workflow to generate a sequence of requests with an empty database and with a populated database in turn.

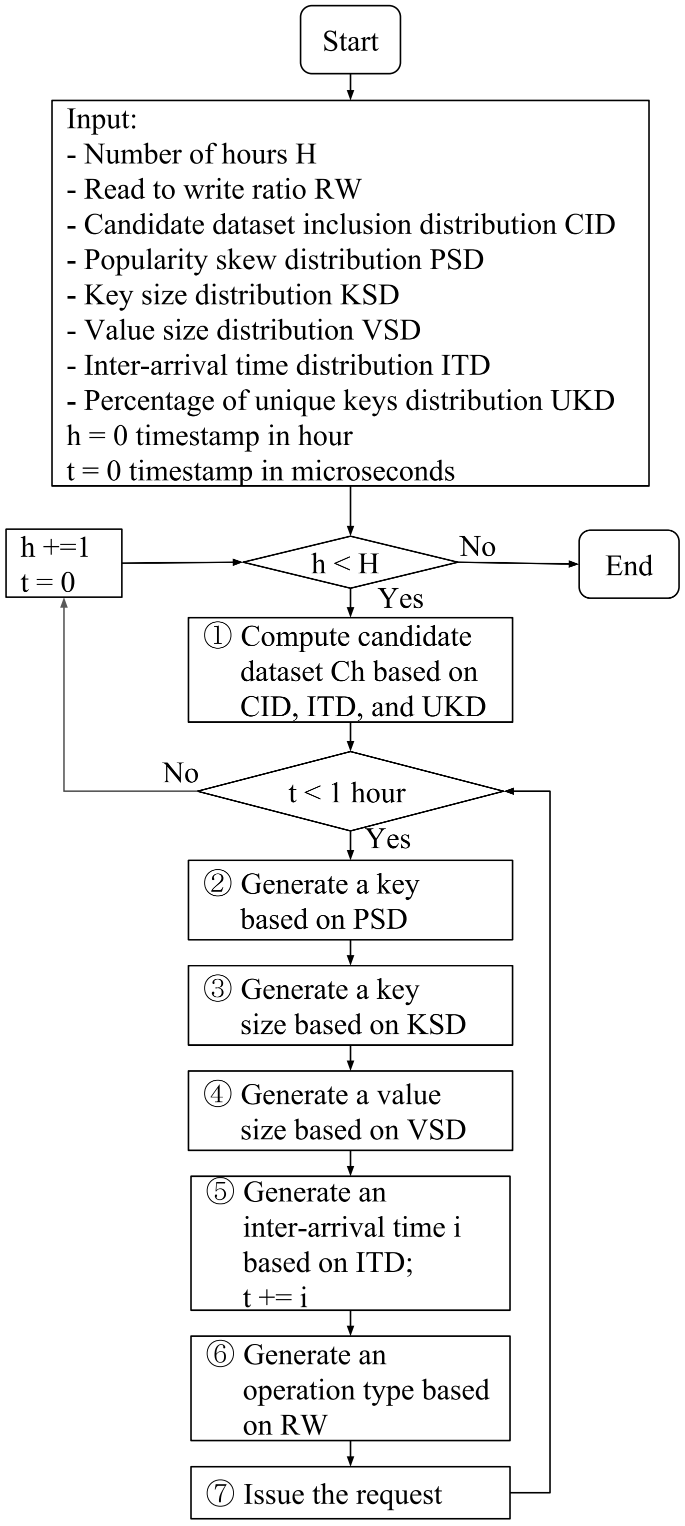
Start
Input:
- Number of hours H
- Read to write ratio RW
- Candidate dataset inclusion distribution CID
- Popularity skew distribution PSD
- Key size distribution KSD
- Value size distribution VSD
- Inter-arrival time distribution ITD
- Percentage of unique keys distribution UKD
h = 0 timestamp in hour
t = 0 timestamp in microseconds
h +=1
t = 0
h < H
No
End
Yes
① Compute candidate dataset Ch based on CID, ITD, and UKD
No
t < 1 hour
Yes
② Generate a key based on PSD
③ Generate a key size based on KSD
④ Generate a value size based on VSD
⑤ Generate an inter-arrival time i based on ITD; t += i
⑥ Generate an operation type based on RW
⑦ Issue the request

Figure 4: Hoagie's workflow with an empty database.

3.1 Workflow with an Empty Database

Fig. 4 shows the workflow for online generation of a sequence of requests with an empty database. The workflow generates a candidate dataset of data items for every hour ①. A

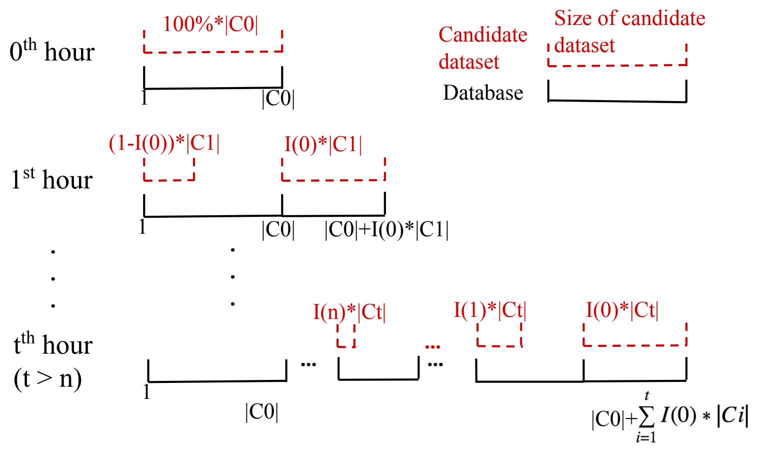

Figure 5: Hoagie's evolving candidate dataset.

candidate dataset is a subset of data items in the database that may be referenced in an hour. Next, it generates a sequence of requests that reference data items in the candidate dataset. The generated working set contains a subset of data items in the candidate dataset of this hour. The workflow terminates when it completes generating requests for the number of hours specified by a user. Fig. 4 highlights the following features of Hoagie:

3.1.1 Candidate Dataset

Hoagie represents a candidate dataset as a list of non-overlapping ranges. A range is represented as $[l, h]$ where l and h are the lowest and highest value of a key. A range includes all values from l to h. The popularity of a key decreases from l to h within the range. The list of ranges is sorted by the lowest value l in ascending order such that the last range represents new keys in a candidate dataset. Newly inserted keys are the most popular. As illustrated in Fig. 5, at the 0^{th} hour, Hoagie represents the candidate dataset C_0 as one range $[1, |C_0|]$ and key 1 is the most popular key.

Hoagie's candidate dataset evolves over time. At the t^{th} hour, the candidate dataset C_t consists of data items in previous hours and a set of new data items.

Hoagie computes the size of candidate dataset $|C_t|$, the number of data items that may be referenced at the t^{th} hour, using the inter-arrival time distribution and the percentage of unique keys of this hour. First, it calculates the total number of requests in an hour as $60 * 60 * 10^6(1\ hour)$ divided by the mean (μs) of the inter-arrival time distribution of this hour. Next, it computes the size of candidate dataset $|C_t|$ as the product of the number of requests R and the percentage of unique keys in this hour. We fix the percentage of unique keys to 20.7% as published by Facebook [4].

Hoagie computes the candidate dataset C_t using an inclusion function $I(x), x \in [0, n]$. $I(x)$ reports the percentage of items from the candidate dataset at a previous hour $t - x$ that are included in the new candidate dataset C_t. $I(0)$ states the percentage of new keys generated in this hour. Hoagie may reference items in the candidate dataset up to n hour(s) ago. As shown in Fig. 5, C_t includes $I(n) * |C_t|$ number of the most popular data items in the candidate dataset C_{t-n}, $I(n-1) * |C_{t-n+1}|$ number of the most popular data items in the candidate dataset C_{t-n+1}, etc. C_t includes all data items in a candidate dataset C_{t-i} if its size $|C_{t-i}|$ is smaller than the required number of data items $|C_t| * I(i)$. Included ranges in two previous candidate datasets may overlap. In this case, they are merged to ensure ranges of the candidate dataset C_t do not overlap. Lastly, Hoagie fills the candidate dataset C_t with new keys.

By default, Hoagie may reference data items created in the past one day (n=24). Hoagie uses $I(0) = 88.5\%, I(1) = 4\%$ based on statistics published by Facebook [4] and speculates the percentage of included candidate datasets of other hours.

Example 1. Assume a user configures Hoagie to include data items in candidate datasets of the past 2 hours, i.e., $I(x), x \in [0, 2]$. This means a candidate dataset must consist of 3 ranges: one for the current hour and two for each of the previous two hours. The user sets $I(0)$=70%, $I(1)$=20%, and $I(2)$=10%. This means the range for the new hour constitutes 70% of its candidate dataset, the range from one hour ago constitutes 20% of the candidate dataset, and the range from two hours ago constitutes 10% of the candidate dataset.

Assume Hoagie completed generating 9 hours of requests and is about to compute the candidate dataset C_{10} at the 10^{th} hour to consist of 4000 data items. Moreover, assume at the 8^{th} hour, C_8 contains three ranges $[101, 200], [401, 600], [1001, 1700]$ with a total of 1000 items. And, at the 9^{th} hour, C_9 contains three ranges $[401, 500], [1001, 1200], [1701, 2400]$ with a total of 1000 items.

Hoagie computes items in C_{10} as follows. First, it includes $I(2)*|W_{10}|$ (400) number of the most popular items in C_8, which is $[1001, 1400]$. Next, it includes $I(1) * |W_{10}|$ (800) number of the most popular items in C_9, which are $[1001, 1100]$ and $[1701, 2400]$. Note the overlap between the ranges $[1001, 1400]$ and $[1001, 1100]$. Hoagie merges them into one $[1001, 1400]$. This produces two ranges, $[1001, 1400]$ and $[1701, 2400]$. Finally, Hoagie constructs the range $[2401, 5300]$, generating 2900 new data items. □

3.1.2 Generating a Request

Hoagie first selects one of the data items from a candidate dataset using the popularity skew distribution ②. New data items in a candidate dataset are the most popular and the popularity of a data item decays over time. We model the popularity skew using the Zipfian distribution.

To model Facebook's key sizes and value sizes, Hoagie is configured as follows. The key-size ③ follows the generalized extreme value distribution with location=30.7984, scale=8.20449 shape=0.078688 [4]. The value-size ④ follows the generalized pareto distribution with location=0, scale=214.476, shape=0.348238 [4].

The workflow generates a timestamp indicating when the request is issued. Hoagie uses the inter-arrival time ⑤ which follows the generalized pareto distribution. The distribution parameters vary across different hours of a day, see Fig. 2. Lastly, it selects the operation type of the request based on the read to write ratio.

3.2 Workflow with a Populated Database

Similar to the workflow with an empty database, the workflow with a populated database also computes a candidate dataset on an hourly basis. However, there are several differences.

First, Hoagie precomputes the database size N based on the benchmark execution time in hours and specifications. Second, Hoagie generates a database of N data items with key sizes and value sizes from specifications. Third, it uses a data item's key-size and value-size in the data store instead of generating them at runtime.

4 Use Cases

An experimentalist may use Hoagie in a simulator or a benchmark to evaluate performance of a design decision. Below, we illustrate two use cases. First, deployment of Hoagie in a simulator to evaluate alternative cache eviction policies. Second, use of Hoagie as the workload generator of the Yahoo! Cloud Serving Benchmark [10] (YCSB). We describe each in turn.

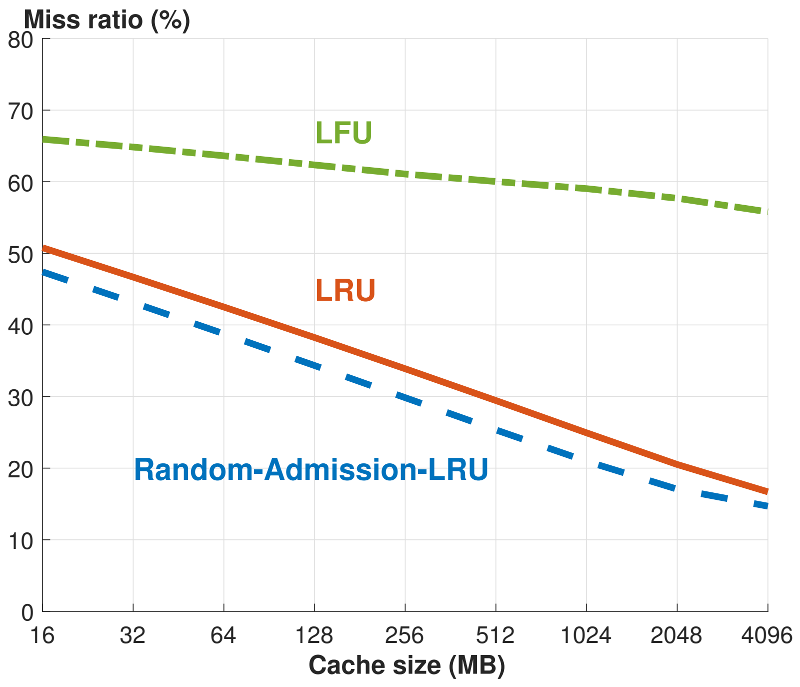

Figure 6: Miss ratio curves of LRU and LFU eviction policy.

4.1 Simulator

Carl et al. [23] show that an eviction policy is sensitive to workload characteristics. A user may evaluate various cache eviction policies, e.g., LRU, LFU, and LHD [7], with Facebook's published specifications by plugging Hoagie in a simulator. A cache entry's size is an item's key-size plus its value-size. To process a read request, the simulator looks up the cache and records a cache hit if the referenced key exists. Otherwise, it records a cache miss and inserts the item into the cache. To process a write request, the simulator implements the write-around policy that deletes the referenced key in the cache. Fig. 6 shows the miss ratio of

LRU, LFU, and LRU extended with an admission policy (Random-Admission-LRU). These results are gathered with different cache sizes using a one-day trace. In these experiments, the database starts empty with no data items. When the cache reaches the maximum capacity, LRU evicts the least recently used item and LFU evicts the least frequently used item. Random-Admission-LRU admits an entry to the LRU cache with 50% probability. We observe the miss ratio of LRU is lower than LFU across all simulated cache sizes. This is because Hoagie models an evolving working set such that the popularity of an item decays over time. A new data item k introduced at h^{th} hour receives a much higher number of requests within this hour compared to $h + 1^{th}$ hour. Ideally, an eviction policy should evict item k at $h + 1^{th}$ hour to make room for new items introduced within this hour. LRU may evict item k since it may become the least recently used item. However, LFU may not evict it due to its high frequency at h^{th} hour. We also observe that Random-Admission-LRU achieves a lower miss ratio than LRU consistently across simulated cache sizes. This is because Zipfian exhibits a long tail distribution. Admitting keys in the long tail pollutes the cache since they may never get referenced again.

4.2 Benchmark

A user may integrate Hoagie into a modular benchmark such as YCSB. YCSB is a synthetic benchmark that models realistic workloads. YCSB requires a user to implement its operations for use with a data store. A user configures a database size, the number of client threads issuing requests, the read to write ratio, and an access pattern to generate a request. A client thread generates a request independent of other threads.

YCSB has two phases: (a) load a database with a fixed number of data items, (b) run a workload to evaluate performance of a data store. Below, we describe the integration of Hoagie into these two phases.

4.2.1 Load

A user specifies the trace duration in hours which determines the database size as described in Section 3.1.1. Then, it may load all data items into the data store. Hoagie generates more than 1 billion keys with a one-day trace. It requires 3.3 TiB storage as the mean data item size is 365 bytes.

The database size grows over time since Hoagie models an evolving working set. A user can bound the database size by decreasing the reference periods of the inclusion function described in Section 3.1.1. By default, Hoagie may reference keys created in the past one day and the database only needs to keep one-day worth of data. Hoagie provides an interface to facilitate a user to delete unused data items. This may not be applicable for some modelled applications, e.g., an online retail application may want to keep the full history of a user's past transactions.

4.2.2 Execution

During the benchmark run, Hoagie uses a single thread to generate requests as described in Section 3.1.2. The required modifications in YCSB are as follows. The Hoagie thread keeps generating requests and stores them in a fixed-sized queue. The queue has a single producer (the Hoagie thread) and multiple consumers (YCSB threads). A YCSB client thread fetches a batch of requests from the queue to process at a time.

5 Related Work

Hoagie is inspired by workload generators of existing benchmarks. Section 4.2 illustrates how YCSB may integrate Hoagie.

Tilman Rabl et al. [19] presents data requirement in parallel data generation, i.e., realistic benchmark specifications and data dependency. Hoagie addresses the data requirements by modeling workload characteristics based on published specifications.

OLTPBench [12] is an extensible testbed for benchmarking relational databases. It generates a sequence of SQL commands for 15 workloads that differ in complexity and system demands, including three workloads derived from real-world applications, i.e., Twitter, Wikipedia, and Epinion. OLTPBench may use Hoagie as its 16th workload which models Facebook's workload characteristics.

YCSB++ [18] extends YCSB to improve understanding of advanced features of a data store, e.g., access control. YCSB++ coordinates YCSB clients to increase load and measure eventual consistency. Hoegie differs in that it focuses on modeling workload characteristics instead of evaluating advanced features of a data store. A user may integrate Hoagie into YCSB++ and extend its coordinator to deploy Hoagie on multiple nodes.

Finally, BG [5] is a benchmark that rates a data store for processing interactive social networking actions. BG's social graph consists of a fixed number of users and a user makes friends with other users. BG may use Hoagie to model an evolving social graph, comments posted on resources, and a working set with temporal locality to generate socialites referencing comments.

6 Future Work

We are extending Hoagie in several ways. First, we are extending its software to be more modular and configurable by changing several hard coded parameters into runtime configuration parameters. To illustrate, generation of data items and requests is hard coded to be every hour. This temporal periodicity can be a runtime configuration parameter. As another example, we envision Hoagie to support different specifications that generate a database and a workload. A challenge is how to provide intuitive configuration files to empower an experimentalist to view all available specifications, select one specification, and switch from one specification to another. Finally, we are investigating how to scale Hoagie across multiple nodes. We intend to quantify the impact of clock skew on the generated requests and the different distributions such as inter-arrival time between requests. The results would enable us to not over-engineer Hoagie. For example, if clock skew in the order of tens of milliseconds has no quantifiable impact then it would eliminate the need for an atomic clock and its software.

7 Acknowledgments

We gratefully acknowledge use of Utah Emulab network testbed [24] ("Emulab") for all experimental results presented in this paper.

References

[1] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben. On the Diversity of Cluster Workloads and Its Impact on Research Results. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 533–546, Boston, MA, 2018. USENIX Association.

[2] M. Arlitt and T. Jin. A Workload Characterization Study of the 1998 World Cup Web Site. *Netwrk. Mag. of Global Internetwkg.*, 14(3):30–37, May 2000.

[3] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. LinkBench: A Database Benchmark Based on the Facebook Social Graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1185–1196, New York, NY, USA, 2013. ACM.

[4] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 53–64, New York, NY, USA, 2012. ACM.

[5] S. Barahmand and S. Ghandeharizadeh. BG: A Benchmark to Evaluate Interactive Social Networking Actions. *CIDR*, January 2013.

[6] S. Barahmand and S. Ghandeharizadeh. D-Zipfian: A Decentralized Implementation of Zipfian. In *Proceedings of the Sixth International Workshop on Testing Database Systems*, DBTest '13, pages 6:1–6:6, New York, NY, USA, 2013. ACM.

[7] N. Beckmann, H. Chen, and A. Cidon. LHD: Improving Cache Hit Rate by Maximizing Hit Density. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 389–403, Renton, WA, 2018. USENIX Association.

[8] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, volume 1, pages 126–134 vol.1, March 1999.

[9] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, M. Marchukov, D. Petrov, L. Puzar, Y. J. Song, and V. Venkataramani. TAO: Facebook's Distributed Data Store for the Social Graph. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 49–60, San Jose, CA, 2013. USENIX.

[10] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[11] P. J. Denning. The Working Set Model for Program Behavior. In *Proceedings of the First ACM Symposium on Operating System Principles*, SOSP '67, pages 15.1–15.12, New York, NY, USA, 1967. ACM.

[12] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proc. VLDB Endow.*, 7(4):277–288, Dec. 2013.

[13] S. Ghandeharizadeh and H. Huang. Gemini: A Distributed Crash Recovery Protocol for Persistent Caches. Technical Report 2018-06 http://dblab.usc.edu/Users/papers/Gemini.pdf, USC Database Laboratory, 2018.

[14] S. Ghandeharizadeh and H. Huang. Hoagie: A Database and Workload Generator using Published Specifications. https://github.com/scdblab/fbworkload/tree/bpod18, 2018.

[15] Q. Huang, P. Ang, P. Knowles, T. Nykiel, I. Tverdokhlib, A. Yajurvedi, P. Dapolito, IV, X. Yan, M. Bykov, C. Liang, M. Talwar, A. Mathur, S. Kulkarni, M. Burke, and W. Lloyd. SVE: Distributed Video Processing at Facebook Scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 87–103, New York, NY, USA, 2017. ACM.

[16] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li. An Analysis of Facebook Photo Caching. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 167–181, New York, NY, USA, 2013. ACM.

[17] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling Memcache at Facebook. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 385–398, Lombard, IL, 2013. USENIX.

[18] 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 *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 9:1–9:14, New York, NY, USA, 2011. ACM.

[19] T. Rabl and M. Poess. Parallel Data Generation for Performance Analysis of Large, Complex RDBMS. In *Proceedings of the Fourth International Workshop on Testing Database Systems*, DBTest '11, pages 5:1–5:6, New York, NY, USA, 2011. ACM.

[20] N. Y. Times. A Face Is Exposed for AOL Searcher No. 4417749. https://www.nytimes.com/2006/08/09/technology/09aol.html, 2018.

[21] N. Y. Times. Facebook Fined in U.K. Over Cambridge Analytica Leak. https://www.nytimes.com/2018/07/10/technology/facebook-fined-cambridge-analytica-britain.html, 2018.

[22] TPC. TPC Benchmark C, 2018.

[23] C. Waldspurger, T. Saemundsson, I. Ahmad, and N. Park. Cache Modeling and Optimization using Miniature Simulations. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 487–498, Santa Clara, CA, 2017. USENIX Association.

[24] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, Dec. 2002.