Toward Workload-Aware State Management in Streaming Systems

Showan Esmail Asyabi
Boston University

Abstract

Modern streaming systems rely on persistent KV stores to perform stateful processing on data streams. Although the choice of the state store is crucial for the system's performance, there has been little research in designing state stores tailored for streaming workloads. Streaming systems use general-purpose KV stores, such as RocksDB, to manage state. Being oblivious to workload characteristics of streaming applications, such stores incur unnecessary overheads. Our methodology to tackle this challenge consists of the following steps. First, we have been conducting a thorough study of streaming state workloads to further the understanding of their characteristics and differences from traditional workloads. Second, we are developing a new benchmark that can faithfully mimic streaming state workloads and enables researchers to easily evaluate alternative store designs. Our long-term goal is to design and develop workload-aware streaming state management to improve the latency and throughput of streaming analytics.[1]

1 Introduction

A streaming system continuously ingests and processes data streams and historical information to perform continuous analytics [9]. The long-running nature of streaming computations often requires maintaining larger-than-memory state. Therefore, streaming systems heavily rely on persistent internal (i.e., embedded) or external stores (e.g., KV stores) to perform stateful processing. [9, 15].

State managers play a vital role in determining the performance of streaming systems. Our preliminary results indicate that streaming operators often access the state store multiple times per incoming event. Thus, state stores need to handle a higher request load than the streaming operators. Nevertheless, state management has received little attention when building modern stream processing systems. The current practice is to use general-purpose stores, such as RocksDB, for maintaining streaming state [11]. However, relying on stores which are oblivious to the streaming workloads characteristics can significantly hamper performance [6, 11].

To evaluate alternative KV stores for a stream processor, system designers currently have no choice but spending time and effort to integrate the store with the reference streaming engine in order to conduct evaluation. Existing streaming benchmarks [1, 2, 5, 17] do not take state management into account. On the other hand, request-driven KV benchmarks, such as YCSB [8], cannot faithfully represent the characteristics of real-world workloads, including streaming workloads [10, 11, 19].

Our goal is to improve the performance of stateful streaming systems by investigating and designing novel methods to store, manage, and query streaming state efficiently. To achieve this goal, we have been working on developing a new benchmarking tool that can generate representative streaming workloads and can be used to evaluate state stores without the need for integration into a streaming engine.

2 Overview of the proposed work

Similar to works in [4, 18, 19], we conduct a comprehensive analysis to characterize streaming workloads. Our preliminary results demonstrate several interesting characters that are overlooked in existing studies.

We will develop a benchmarking tool for streaming state management that allows configuring a set of streaming-specific parameters that capture streaming application behavior. To the best of our knowledge, this will be the first benchmark for streaming state managers. We hope our tool will provide a foundation for the evaluation of existing and future state store designs and their suitability to streaming workloads.

We will finally leverage our findings in the previous steps to design and build novel methods for internal and external state management in streaming systems. We hope our de-

[1] PhD advisor: Vasia Kalavri and Azer Bestavros- Boston University
PhD start date : Fall 2018
Expected submission date of the PhD thesis : Fall 2022

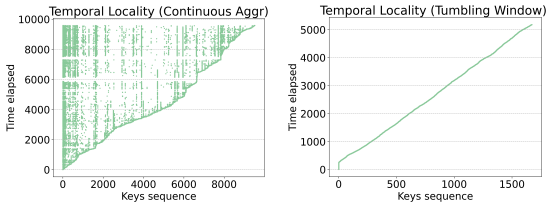

Figure 1: The two operators have the same read&write ratio. However, unlike the continuous aggregation operator, the tumbling widow operator has a high temporal locality.

sign awareness of state workload characteristics will notably increase the performance of streaming systems running on clouds and local clusters.

3 Preliminary Results

We have instrumented the Apache Flink streaming system to monitor the workload it imposes on the state manager (RocksDB). We used Google's Borg cluster workload traces [16] to design different Flink applications that showcase the state access workload of different operators. We study several standard streaming operators, such as windows, joins, and continuous aggregations. Our analysis so far demonstrates that streaming applications impose state access workloads with unique characteristics.

We analyze the state workload of each operator based on the **access pattern**, **temporal locality**, **working set size**, **hit rates** for different cache sizes, **key popularity distribution**, and **access amplification**. Temporal locality shows how closely a current key will be accessed in the future. Hit rate demonstrates the ratio of hits if the keys are being cached in caches with different sizes. The working set size shows a cache size the achieves the highest possible hit rate. The Key popularity shows the popularity of keys compared to one another. We define access amplification as the number of interactions an operator has with the state manager per each incoming event.

Our fine-grained analysis uncovers several unique characteristics. Most of the operators have high temporal locality. As shown in Figure 1, tumbling windows operator merely access **one** key until the key is deleted and then they processed with a new key. Current benchmarks cannot capture the properties of streaming state workloads. For example, when configuring YCSB we can control the ratio of reads to writes but we cannot control the temporal locality of keys. Operators in Figure 1 would be represented by the same workload type (50% read & 50% write), even though they are drastically different in terms of temporal locality and access sequence.

Moreover, we have identified unique characters that can guide better store designs. As an example, RocksDB performs significant computations and IO to optimize the read path [12]. We believe this overhead can be eliminated if the store is aware of the operator's temporal locality, especially in cases when keys will never be looked up.

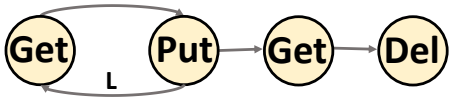

Figure 2: Tumbling window state machine

4 Work to be Done

Our current plan for building the benchmarking tool is to represent each operator as a temporal finite-state machine, where states represent operations (e.g., get and put). Figure 2 shows the corresponding state machine of a tumbling window, where L is the window's length. The challenge we face here is to design state machines for more complicated operators such as session windows.

We will use the real-world traces collected in the first step and our re-player to compare how closely our benchmark can mimic the real-world traces. We will then evaluate existing KV stores using our benchmark to showcase current KV stores' capabilities supporting different streaming systems operators. We will finally use the lessons learnt as guidance to design and build novel stores, aware of the characteristics of streaming workloads. A major challenge to achieve this goal is that streaming operators have different requirements.

5 Related Work

Workload characterization. [18] and [4] perform comprehensive analyses to characterize in-memory KV stores used as cache nodes in Twitter and Facebook, respectively. [19] characterizes workloads of three typical RocksDB production use-cases at Facebook. In similar spirit, we analyze RocksDB workloads when used for stateful streaming.

Benchmarking. YCSB [8] is a widely used benchmark which can be tuned to generate a variety of real-world workloads. However, it has been shown that YCSB [19] cannot represent important aspects, such key-space locality [10]. Our preliminary results indicate that YCSB cannot mimic the main characters of streaming state workloads such as temporal locality.

Managing states in streaming systems. RocksDB [14] is widely used by open-source systems such as Apache Spark Structured Streaming [3], Apache Flink [7], and Apache Samza [13]. Although RocksDB offers solid performance and many capabilities such as incremental checkpointing, recent work [6, 11] has shown that using stores oblivious to operators characteristics adversely impacts the performance of streaming systems. Our vision is to address this challenge by designing state managers aware of the characteristics of streaming systems.

References

[1] Nexmark Benchmark Suite. https://beam.apache.org/documentation/sdks/java/testing/nexmark/. Last access: February 2021.

[2] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: a stream data management benchmark. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 480–491, 2004.

[3] Michael Armbrust, Tathagata Das, Joseph Torres, Burak Yavuz, Shixiong Zhu, Reynold Xin, Ali Ghodsi, Ion Stoica, and Matei Zaharia. Structured streaming: A declarative api for real-time applications in apache spark. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 601–613, New York, NY, USA, 2018. Association for Computing Machinery.

[4] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike 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, page 53–64, New York, NY, USA, 2012. Association for Computing Machinery.

[5] Yahoo Streaming Benchmark. https://github.com/dataArtisans/yahoo-streaming-benchmark. Last access: February 2021.

[6] Matthew Brookes, Vasiliki Kalavri, and John Liagouris. Faster state management for timely dataflow. In *Proceedings of Real-Time Business Intelligence and Analytics*, BIRTE 2019, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. State management in apache flink®: Consistent stateful distributed stream processing. *Proc. VLDB Endow.*, 10(12):1718–1729, August 2017.

[8] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 143–154, New York, NY, USA, 2010. Association for Computing Machinery.

[9] Marios Fragkoulis, Paris Carbone, Vasiliki Kalavri, and Asterios Katsifodimos. A survey on the evolution of stream processing systems. *arXiv preprint arXiv:2008.00842*, 2020.

[10] Eran Gilad, Edward Bortnikov, Anastasia Braginsky, Yonatan Gottesman, Eshcar Hillel, Idit Keidar, Nurit Moscovici, and Rana Shahout. Evendb: Optimizing key-value storage for spatial locality. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Vasiliki Kalavri and John Liagouris. In support of workload-aware streaming state management. In *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*. USENIX Association, July 2020.

[12] Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Wisckey: Separating keys from values in ssd-conscious storage. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 133–148, Santa Clara, CA, February 2016. USENIX Association.

[13] Shadi A. Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H. Campbell. Samza: Stateful scalable stream processing at linkedin. *Proc. VLDB Endow.*, 10(12):1634–1645, August 2017.

[14] RocksDB. https://rocksdb.org/. Last access: February 2021.

[15] Quoc-Cuong To, Juan Soto, and Volker Markl. A survey of state management in big data processing systems. *The VLDB Journal*, 27(6):847–872, December 2018.

[16] Borg Cluster Workload Traces. https://github.com/google/cluster-data. Last access: February 2021.

[17] Pete Tucker, Kristin Tufte, Vassilis Papadimos, and David Maier. NEXMark—A Benchmark for Queries over Data Streams. Technical report, OGI School of Science & Engineering at OHSU, 2002.

[18] Juncheng Yang, Yao Yue, and K. V. Rashmi. A large scale analysis of hundreds of in-memory cache clusters at twitter. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 191–208. USENIX Association, November 2020.

[19] zhichao Cao, Siying Dong, Sagar Vemuri, and David H.C. Du. Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 209–223, Santa Clara, CA, February 2020. USENIX Association.