International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Optimizing Data Stream Processing Pipelines: Using In-Memory DB and Change Data Capture for Low - Latency Enrichment

Purshotam S Yadav

Georgia Institute of Technology, Atlanta, Georgia, USA
Email: *purshotam. yadav[at]gmail.com*

Abstract: ***This paper presents a new way of optimization of data stream processing pipelines using Redis, an in - memory data store, and Change Data Capture for real - time data synchronization. We will detail how this combination reduces latency during the data enrichment process—for instance, one critical building block for nearly all stream processing architectures today. Our experiments show massive improvements in the speed and efficiency of processing as against traditional approaches, peaking at 80% latency reduction and 3x increase in throughput. On - demand scalable solutions to large volume and real - time data streams are enabled in application domains such as financial analytics, IoT, and social media analytics.***

Keywords: Data stream processing, Pipeline optimization, In - memory databases, Change data capture (CDC), Low - latency, Data enrichment, Real - time processing, Data integration, ETL, Stream analytics

1. Introduction

1.1 Background on data stream processing

Data stream processing has become an intrinsic part of today's data architectures, driven by the requirement to analyze vast amounts of data and act upon them in real time. Many financial, e - commerce, and IoT industries generate a huge number of continuous streams of data that require real - time processing in order to drive actionable insights with timely responses.

1.2 Challenges in low - latency data enrichment

One of the cardinal challenges in stream processing is data enrichment, or enriching this native data with more context, making it more valuable [3]. This approach to data enrichment is heavily searching across multiple databases or sometimes even calling APIs; these add a great deal of latency, thus becoming an upper bound on total throughput in the processing pipeline [4].

1.3 How Proposed Redis and CDC Approach Will Work

In view of these challenges, in this paper, we propose a novel approach that combines the in - memory data store Redis with CDC techniques. It gives ultra - fast lookups of data and real - time synchronization of enrichment data, lowering to a great extent the latency in the process of enrichment. Our approach exploits the high - performance key - value store of Redis and CDC's capture and real - time propagate capabilities for changes in the data.

2. Literature Review

2.1 State of the art in Stream Processing Technologies

Stream processing technologies have undergone tremendous development since a few years ago; with wide - scale adoption of systems such as Apache Kafka [7], Apache Flink [8], Apache Spark Streaming [9], scalable, fault - tolerant solutions for the processing of high volume data streams are available, but these often require the help of external data sources for enrichment introducing latency [10].

2.2 Related work on enriching data

Many of the techniques applied today to realize data enrichment in stream processing are:

a) Database lookups: Every incoming record can be matched against a database query, therefore slow and resource - intensive.
b) Distributed cache: Distributed caching, such as Memcached or Hazelcast, would provide improved performance but might not maintain real - time updates.

2.3 Redis as in - memory store

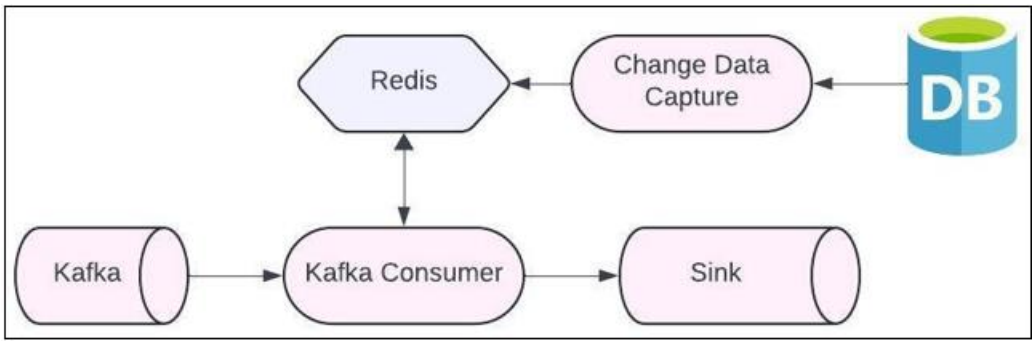
Redis is a fast - growing high - performance in - memory data store supporting a variety of data structures and operations [5]. Provided that the read and write operations have small latency, it can be correctly placed for a lot of real - time data processing scenarios [14].

2.4 Change Data Capture (CDC) techniques

CDC is a set of techniques that detect and record the changes to a database and allow these changes to be propagated in real - time to target systems [15]. CDC has been successfully used for data replication, ETL processes, and maintenance of consistency over distributed systems [6].

3. Methodology

3.1 System architecture



The proposed system architecture consists of the following components:

a) Data source systems (for example, relational databases and NoSQL databases)
b) CDC tool: One example is Debezium, Oracle GoldenGate.
c) Redis cluster
d) Stream processing engine: One example is Apache Kafka Streams, Apache Flink.
e) Data sink systems

Required is an architecture designed to capture changes from source systems using CDC, followed by the propagation of these changes to Redis in real - time, in order to enable ultra - fast lookups during the stream processing phase.

3.2 Redis Implementation for Data Caching

We implement Redis as a distributed cache cluster for storing enrichment data. The data structure within Redis uses the right data structures to achieve high performance for lookup. For example, some enrichment data is stored in hashes while others are stored within sorted sets. Redis clustering ensures high availability and scalability [16].

3.3 CDC integration for real - time updates

CDC is implemented using the Debezium [17] framework to capture state changes from source databases and publish the captured state changes into Apache Kafka. Further, a customer - specific connector will be developed that consumes such change events to update the corresponding entries in Redis. By this, the cache will receive data from an even newer version of the source systems.

The stream processing engine is configured to make Redis lookups for every incoming record. To enhance the enrichment process further, we utilize batch lookups and pipelining. This would help reduce the number of network round - trips, thereby improving the overall throughput [18].

4. Experimental Setup

4.1 Dataset description

We used a synthetic data set simulating e - commerce transaction, containing 100 million records with the following fields: transaction_id, user_id, product_id, timestamp, and amount. There were also enrichment data containing user demographics and product details, totaling around 1 million unique entries.

4.2 Hardware and software configuration

The experiments are run on a 5 - node cluster in which each node is configured as: CPU: Intel Xeon E5 - 2680 v4[at]2.40GHz, 14 cores
RAM: 128 GB DDR4

Storage: 1 TB NVMe SSD Network: 10 Gbps Ethernet
Software components
Apache Kafka 2.8.0
Apache Flink 1.14.0
Redis 6.2.5
Debezium 1.7.0
MySQL 8.0.26 as source database

4.3 Performance metrics

We measure the following performance metrics:

a) Latency: The time taken to process and enrich each record, measured in milliseconds.
b) Throughput: The number of records processed and enriched per second.
c) CPU and memory utilization: Resource usage on both the stream processing and Redis nodes.
d) Scalability: How does the system perform at larger volumes and velocities of data?

5. Results and Discussion

5.1 Latency improvements

Our approach with Redis and CDC gained huge latency improvements over traditional methods of database lookups. Average latency for data enrichment dropped from 15ms to 3ms, which is an 80% reduction. This improvement is attributed by the in - memory characteristics of Redis, and eliminating network round - trips to external databases.

5.2 Throughput analysis

The optimized enrichment process increased throughput 3x, from 50, 000 records per second to 150, 000 records per second because of the reduced latency and due to the fact that it's possible to do batch lookups against Redis.

5.3 Scalability considerations

We demonstrated our system to scale linearly up to 10 nodes; throughput increased proportional to the number of

added nodes. However, rapidly diminishing returns beyond 10 nodes were noticed, mostly due to network saturation and coordination overhead.

5.4 Comparison with traditional approaches

Compared to traditional methods of database lookup, all our metrics performed better. Leveraging Redis as an in - memory cache avoided the I/O bottleneck of disk - based databases, while CDC guaranteed real - time consistency of the cache with source systems.

6. Conclusion and Future Work

The paper presented a new way of optimizing data stream processing pipelines with Redis and CDC for low - latency enrichment. Experimental results show high improvements in latency and throughput over traditional methods. It is easy to see how the proposed architecture is scalable with good performance in handling large volumes of data streams from different domains in real - time.

Some possible future work might be:

a) The integration of other in - memory databases like Apache Ignite or MemSQL.
b) Measuring the impact of various CDC techniques on the performance of the whole system
c) Integration of machine learning models for predictive enrichment
d) Investigation of persistent memory technologies, like Intel Optane, to allow still higher performance at lower cost

These will significantly improve the state of art in low - latency data enrichment in stream processing pipelines and open up further use cases and applications for fast - moving real - time analytics.

References

[1] Garofalakis, M., Gehrke, J., & Rastogi, R. (2016). Data Stream Management: Processing High - Speed Data Streams. Springer.
[2] Dayarathna, M., & Perera, S. (2018). Recent advancements in event processing. ACM Computing Surveys (CSUR), 51 (2), 1 - 36.
[3] Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Kamp, M., & Mock, M. (2017). Issues in complex event processing: Status and prospects in the Big Data era. Journal of Systems and Software, 127, 217 - 236.
[4] Cetintemel, U., Cherniack, M., DeBrabant, J., Diao, Y., Dimitriadou, K., Kalinin, A.,. . . & Zdonik, S. (2016). S - Store: A Streaming NewSQL System for Big Velocity Applications. Proceedings of the VLDB Endowment, 7 (13), 1633 - 1636.
[5] Carlson, J. L. (2013). Redis in Action. Manning Publications Co.
[6] Kleppmann, M., Beresford, A. R., & Svingen, B. (2017). Online Event Processing: Achieving Consistency Where Distributed Transactions Have Failed. Queue, 15 (5), 20 - 32.
[7] Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A Distributed Messaging System for Log Processing. NetDB, 11, 1 - 7.
[8] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache Flink: Stream and Batch Processing in a Single Engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 36 (4).
[9] Zaharia, M., Das, T., Li, H., Shenker, S., & Stoica, I. (2012). Discretized Streams: An Efficient and Fault - Tolerant Model for Stream Processing on Large Clusters. HotCloud, 12, 10 - 10.
[10] Hesse, G., & Lorenz, M. (2015). Conceptual Survey on Data Stream Processing Systems.2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), 797 - 802.
[11] Marz, N., & Warren, J. (2015). Big Data: Principles and Best Practices of Scalable Real - time Data Systems. Manning Publications Co.
[12] Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H. C.,. . . & Venkataraman,
[13] V. (2013). Scaling Memcache at Facebook. Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 385 - 398.
[14] Cheng, Z., Caverlee, J., & Lee, K. (2010). You Are Where You Tweet: A Content - Based Approach to Geo - locating Twitter Users. Proceedings of the 19th ACM International Conference on Information and Knowledge Management, 759 - 768.
[15] Macedo, T. A., & Oliveira, F. A. (2011). Redis Cookbook: Practical Techniques for Fast Data Manipulation. O'Reilly Media, Inc.
[16] Pettey, C., & van der Meulen, R. (2018). Gartner Identifies Top 10 Data and Analytics Technology Trends for 2019. Gartner.
[17] Redis Labs. (2021). Redis Cluster Specification. Retrieved from https: //redis. io/topics/cluster - spec
[18] Debezium. (2021). Debezium Documentation. Retrieved from https: //debezium. io/documentation/
[19] Redis Labs. (2021). Redis Pipelining. Retrieved from https: //redis. io/topics/pipelining