Rethinking Message Brokers on RDMA and NVM

Hendrik Makait
hendrik.makait@campus.tu-berlin.de
Technische Universität Berlin

ACM Reference Format:
Hendrik Makait. 2020. Rethinking Message Brokers on RDMA and NVM. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3318464.3384403

1 INTRODUCTION

Over the last years, message brokers have become an important part of enterprise systems. As microservice architectures gain popularity and the need to analyze data produced by these services grows, companies increasingly rely on message brokers to orchestrate the flow of events between different applications as well as between data-producing services and streaming engines that analyze the data in real-time.

Current state-of-the-art message brokers such as Apache Kafka [9] or Apache Pulsar [5] were designed for slow networks and disk-based storage. Consequently, they avoid network traffic, and small or random writes to persistent storage.

Recent advancements in modern hardware change how we can design distributed systems and as a result, we introduce the design of a message broker that leverages the capabilities of remote direct memory access (RDMA) and non-volatile memory (NVM) to improve on the weaknesses of existing message brokers and further scale these systems. Specifically, our architecture and protocol leverage the high bandwidth and low latency of RDMA and combine those with the byte-addressability and high bandwidth of NVM for guaranteed and in-order message delivery with high throughput. Our contributions are as follows: (1) We propose a decoupled message broker architecture and accompanying protocol that are designed to guarantee message delivery and message ordering at high throughput (Section 3). (2) We demonstrate the potential for improvement of current message broker systems with respect to messaging guarantees at the example of Apache Kafka (Section 4).

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06.
https://doi.org/10.1145/3318464.3384403

2 BACKGROUND

In this section, we introduce Apache Kafka as an example of state-of-the-art message brokers and describe several developments in modern hardware that we use in our architecture.

Apache Kafka [9] is a common choice of message broker used with streaming engines. It creates a distributed, replicated message queue and persists the messages on secondary storage. Persisting data enables consumers to replay data if needed and provides configurable guarantees around message delivery. Stream processing engines such as Apache Flink [4] rely on this feature to ensure exactly-once semantics in the case of task failure. Internally, Kafka stores each of its partitions as an independent append-only log that is written to the page buffer. This design allows Kafka to achieve high throughput and low latency at the cost of its messaging guarantees, as we demonstrate in Section 4.

InfiniBand (IB) is a network communications standard used in modern data centers that offers bandwidths of up to 600 Gbit/s (1.2 Tbit/s by 2020) and latencies below $2\mu s$ [7, 15].

Remote Direct Memory Access (RDMA) can be used as a communications stack in IB networks. It enables applications to directly access memory on a remote machine with little to no involvement of the remote CPU. The application can directly transfer data from user-space bypassing the kernel. It offers two different APIs: One-sided verbs such as read and write operations are executed without any involvement of the remote CPU. Two-sided verbs enable RPC calls without the overhead of TCP/IP-based communication but with the involvement of the remote CPU. Research has shown that distributed systems such as databases benefit from IB if subjected to significant architectural changes [11, 15]. By decoupling computation and storage via a network-attached memory architecture, Zamanian et al. [14] have significantly improved the transactional throughput of a distributed DBMS.

Non-Volatile Memory (NVM) is a new class of memory that bridges the gap between DRAM and flash-based SSD. It combines the byte-addressable access by the CPU known from DRAM with persistent writes offered by SSD. Its access latency and bandwidth lie within an order of magnitude of DRAM with a capacity of up to 512 GB, which is 4x higher than available DRAM [8, 13]. NVM can be used as an alternative to disks or flash-based SSDs for durable storage. Several approaches have been proposed to use it as part of a storage hierarchy for databases [1, 2, 10, 12]. In particular, Huang et al. [6] propose NVM-based logging for transactions.

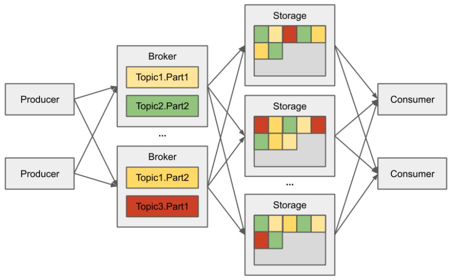

Figure 1: Our architecture separates partition handling from data storage on dedicated nodes.

3 RETHINKING THE ARCHITECTURE

In Figure 1, we present the architecture, which separates the computation-intensive partition handling on brokers from message storage on storage nodes.

Storage. Partitions are split into individual segments that are stored across all storage nodes. Conceptually, these nodes form one large data region, in which brokers can allocate space for individual segments. As a consequence, the user can scale the system by adding another storage node without moving any data. Moreover, incoming data is written into NVM where it is directly persisted. This guarantees message delivery without the additional overhead of fragmented and small writes to secondary storage or of keeping a dedicated journal. Given that our protocol minimizes the involvement of the storage node's CPU, a node may be co-located with compute-heavy workloads without significantly affecting their performance. Nonetheless, storage nodes need to track the usage of their memory segments, provide new ones for brokers and free outdated ones. In our architecture, NVM can either store all data or serve as an intermediate layer combined with SSDs or disks. Both variants and their trade-offs should be evaluated in future work.

Partition handling. For each partition, the write/read requests are handled by a designated broker. Using the RDMA-based protocol we introduce later, the broker provides producers and consumers with the location where data can be read from or written to on a storage node. Further, the broker is responsible for allocating new segments that can be written on storage nodes, handling timeouts by producers and inconsistencies between partitions. Since all data required by the broker are stored on storage nodes, partition handling can be moved to a different broker without much effort.

Co-location. While all data are conceptually stored in remote storage, the co-location of data with their users is a viable optimization. As an example, to reduce latency, the data structures managing a partition could be co-located with their respective broker.

Protocol. For efficient communication within the system, the protocol we propose relies on RDMA verbs to move data between producers, brokers, storage, and consumers. As mentioned before, the protocol minimizes the active involvement of both broker and storage nodes in order to avoid bottlenecks. The protocol contains multiple steps for both writing and reading data, which we outline in the following.

Writing. To write messages to a partition, a producer first reserves a memory area within the currently written segment (and its replicas) using two-sided RDMA verbs. To ensure message ordering, these requests must be issued sequentially to the broker. Given the request size, however, we expect the latency to be small enough to not become a bottleneck. Binnig et al. [3] showed a latency of around 1μs for send/recv verbs with message sizes below 256B. In the next step, the producer uses one-sided RDMA writes to store the messages directly in NVM at the locations received in the previous step. Since these writes are byte-addressable, they can be performed in parallel and out-of-order without changing the order of the messages in the partition. Finally, the producer commits its write to the broker to ensure consistency.

Reading. Analogous to writing, a consumer initially requests a memory location on a storage node from which it can read a sequence of messages given a message offset. This request uses two-sided RDMA verbs, which allows the broker to perform load balancing or trigger a storage node to load a segment into NVM. The consumer then performs a one-sided RDMA read to load the data into its own memory without the involvement of the storage node.

4 EVALUATION AND DISCUSSION

To demonstrate the potential for improving performance with delivery guarantees, we conducted an experiment at the example of Apache Kafka. In this experiment, a single broker hosted one topic with 24 partitions on three SSDs. For each partition, one dedicated producer generated data at the maximum rate. Without any guarantees, the write throughput of 1,031 MB/s is within 20% of the maximum for the SSDs of 1,225 MB/s. Guaranteed message order reduces this by 25%, and guaranteed delivery, as well as the combination of both, cause a decrease by a factor of more than 250x. By enabling parallel message transfer and using byte-addressable writes to NVM with high throughput, we expect our approach to avoid such a degradation in performance.

In conclusion, we propose a message broker architecture that decouples partition handling and storage for improved scaling of storage and load balancing between different storage nodes or brokers. Its protocol ensures message ordering even in the event of retries for parallel messages by utilizing byte-addressable RDMA writes. To ensure message delivery, producers write messages to NVM. Finally, direct transfer of message payloads to or from storage nodes using one-sided RDMA-verbs reduces the CPU overhead on storage nodes and the involvement of brokers. In the future, we plan to implement this architecture and evaluate it in detail.

REFERENCES

[1] Joy Arulraj and Andrew Pavlo. 2017. How to Build a Non-Volatile Memory Database Management System. In *Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17*. ACM Press, Chicago, Illinois, USA, 1753–1758. https://doi.org/10.1145/3035918.3054780

[2] Joy Arulraj, Andrew Pavlo, and Subramanya R. Dulloor. 2015. Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*. ACM Press, Melbourne, Victoria, Australia, 707–722. https://doi.org/10.1145/2723372.2749441

[3] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. 2016. The end of slow networks: it's time for a redesign. *Proceedings of the VLDB Endowment* 9, 7 (March 2016), 528–539. https://doi.org/10.14778/2904483.2904485

[4] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink™: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015), 12.

[5] The Apache Software Foundation. 2019. Apache Pulsar. https://pulsar.apache.org/

[6] Jian Huang, Karsten Schwan, and Moinuddin K. Qureshi. 2014. NVRAM-aware logging in transaction systems. *Proceedings of the VLDB Endowment* 8, 4 (Dec. 2014), 389–400. https://doi.org/10.14778/2735496.2735502

[7] InfiniBand Trade Association. 2019. InfiniBand Roadmap - Advancing InfiniBand. https://www.infinibandta.org/infiniband-roadmap/

[8] Intel Corporation. 2019. Intel® Optane™ DC Persistent Memory Product Brief. https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-dc-persistent-memory-brief.html

[9] Jay Kreps, Neha Narkhede, and Jun Rao. 2011. Kafka: A Distributed Messaging System for Log Processing. 1–7.

[10] Steven Pelley, Thomas F. Wenisch, Brian T. Gold, and Bill Bridge. 2013. Storage management in the NVRAM era. *Proceedings of the VLDB Endowment* 7, 2 (Oct. 2013), 121–132. https://doi.org/10.14778/2732228.2732231

[11] Animesh Trivedi, Patrick Stuedi, Jonas Pfefferle, Radu Stoica, Bernard Metzler, Ioannis Koltsidas, and Nikolas Ioannou. 2016. On the [ir] relevance of network performance for data processing. In *8th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 16)*.

[12] Alexander van Renen, Viktor Leis, Alfons Kemper, Thomas Neumann, Takushi Hashida, Kazuichi Oe, Yoshiyasu Doi, Lilian Harada, and Mitsuru Sato. 2018. Managing Non-Volatile Memory in Database Systems. In *Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18*. ACM Press, Houston, TX, USA, 1541–1555. https://doi.org/10.1145/3183713.3196897

[13] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. 2019. Persistent Memory I/O Primitives. *arXiv:1904.01614 [cs]* (April 2019). http://arxiv.org/abs/1904.01614 arXiv: 1904.01614.

[14] Erfan Zamanian, Carsten Binnig, Tim Harris, and Tim Kraska. 2017. The end of a myth: distributed transactions can scale. *Proceedings of the VLDB Endowment* 10, 6 (Feb. 2017), 685–696. https://doi.org/10.14778/3055330.3055335

[15] Steffen Zeuch, Bonaventura Del Monte, Jeyhun Karimov, Clemens Lutz, Manuel Renz, Jonas Traub, Sebastian Breß, Tilmann Rabl, and Volker Markl. 2019. Analyzing efficient stream processing on modern hardware. *Proceedings of the VLDB Endowment* 12, 5 (Jan. 2019), 516–530. https://doi.org/10.14778/3303753.3303758