Caches ~~Rule~~ Replicate Everything Around Me

Michael Abebe, Salesforce

In modern software development, *caches* rule everything around us [1]. For example, consider the basic architecture of a modern web application: featuring web servers, a service or application tier, and a database layer [3, 2, 17]. Each of these components use caches to reduce the latency of accessing data by moving data storage closer to its destination than the source. In this paper, we focus on caches used by applications to reduce the latency of accessing distributed data systems. Such caches are typically stateless and thus allow for independent horizontal scaling from the database. Thus, adding a cache is often the first approach used when addressing database performance problems [19]. However, caching can also introduce several *challenges* related to: (i) the consistency (or coherence) of cached data, (ii) cold starts (or empty caches) resulting in a thundering herd of load on the underlying database, and (iii) selecting policies for data eviction such as access frequency or recency [2, 3, 19, 11].

An alternative view is that data caching is just another form of data *replication*. Replication of data in distributed systems is a well-studied problem, and existing protocols define the set of data items to be replicated, where the replicas should be placed, and which updates should be propagated to replicas and when [24, 14]. Caching is a form of *adaptive* replication that uses specific policies to select which data to replicate (e.g., access frequency or recency) and policies to ensure data consistency or freshness, such as write-through caching or look-aside caching [7, 14, 1].

Given this framing of caching as replication and the goals of providing low-latency data access with application-specific consistency semantics, an alternative approach to system design is to *integrate* caches directly into the underlying distributed data system. Doing so allows the system to leverage its existing replication and consistency protocol by extending its model of what data is replicated to data contained in the cache. That is, data in the cache is treated as yet another replica by the data system and maintained by the system. What was the cache becomes just another resource managed by the system as it desires. While such an approach may eschew traditional software design principles, such as separation of concerns, an analogous design choice was made by traditional disk-backed database systems. Such databases manage their disk-backed page directly using database buffer caches and define concurrency control and recovery protocols to ensure correctness instead of relying on traditional operating system primitives [21]. Databases make this design trade-off because of the significant performance advantage that can be gained because of knowledge that the database has over how data will be accessed, for instance, pre-fetching pages based on query plans [22].

How might a distributed data system integrate a cache directly into its architecture? Or alternatively, how might such systems utilize adaptive replication in an environment where server resources are heterogeneous? As a simple first example, consider a distributed block storage system like HDFS [20] that replicates data for fault tolerance but is also frequently deployed with a separate block cache [16]. If the distributed storage system also controlled what blocks were cached and when [18], then it could monitor the workload to predictively place blocks likely to be accessed in the future in the cache. Additionally, the system could place blocks with moderate access frequency in the cache at the expense of one persistent replica provided the durability requirements continue to be met. A second, more complex example is a distributed database that uses quorum reads and writes to manage the consistency of data partition [10, 12]. Careful placement of data partitions such that read quorums for data partitions can be satisfied solely from the caching tier for the most frequently read data may improve system performance. A third deeper integration of caching is to the cached data as a form of a materialized view [13]. That is, if an application caches the results of a query, then with the cache under the database control, it can atomically update the cache transactionally with updates that alter the materialized data in the cache. If such a design is also integrated with the isolation semantics, for example, query results stored in the materialized view/cache execute as of a snapshot, then they can be used for subsequent transactions if they satisfy the snapshot semantics [15].

How can a system make decisions about what to adaptively cache to integrate a cache into the system? As a framing exercise, we consider adaptive replication as a constraint problem that must be satisfied [7, 14]. Given a set of server resources (e.g., amount of memory, disk, CPU, network bandwidth of each server), place the data items on these resources such that they satisfy fault tolerance criteria (e.g., each data item must have at least three replicas), and optimize for a performance metric (e.g., minimal tail latency or maximize throughput) while satisfying the required consistency level (e.g., provide strong consistency). Rather than solving this as a constrained optimization problem, the system should utilize machine learning and reinforcement learning techniques to make adaptive replication decisions in an online manner based on the workload [9, 23].

In summary, we argue that *distributed databases should directly manage cached data and treat the cache as yet another replica*. Doing so offers opportunities for improved system performance and can leverage the existing replication protocol. Finally, using machine learning to drive decisions about what to replicate offers a mechanism to make adaptive decisions about what to replicate, which allows the system to respond to the workload, potentially even predictively.

[1]Wu-Tang Clan C.R.E.A.M.: Cash Rules Everything Around Me `https://youtu.be/t3NvfDnz3Ds`

About the Author:

Michael is a researcher and Senior Member of the Technical Staff at Salesforce, where he works on distributed database systems, with a particular focus on transaction processing and replication. His interests are at the intersection of distributed systems and databases and applying machine learning to database systems. In 2022, he earned a Ph.D. in Computer Science from the University of Waterloo. His thesis was titled *Adaptive Data Storage and Placement in Distributed Database Systems* [4], and his research examined distributed commit protocols [6], adaptive replication and partitioning [7], hybrid OLTP and OLAP database systems [8], online prediction in databases [9], and distributed storage systems [5]. He also co-delivered a tutorial on adaptive replication and partitioning in distributed data systems [14]. More information, including links to all publications, can be found on his website: `https://cs.uwaterloo.ca/~mtabebe/`

References

[1] Cache-aside pattern - azure architecture center.

[2] Caching best practices - amazon web services.

[3] Caching guidance - azure architecture center.

[4] M. Abebe. *Adaptive Data Storage and Placement in Distributed Database Systems*. PhD thesis, University of Waterloo, 2022.

[5] M. Abebe, K. Daudjee, B. Glasbergen, and Y. Tian. Ec-store: Bridging the gap between storage and latency in distributed erasure coded systems. In *2018 IEEE 38th international conference on distributed computing systems (ICDCS)*, pages 255–266. IEEE, 2018.

[6] M. Abebe, B. Glasbergen, and K. Daudjee. Dynamast: Adaptive dynamic mastering for replicated systems. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1381–1392. IEEE, IEEE, 2020.

[7] M. Abebe, B. Glasbergen, and K. Daudjee. Morphosys: Automatic physical design metamorphosis for distributed database systems. *Proceedings of the VLDB Endowment*, 13(13):3573–3587, 2020.

[8] M. Abebe, H. Lazu, and K. Daudjee. Proteus: Autonomous adaptive storage for mixed workloads. In *Proceedings of the 2022 International Conference on Management of Data*, pages 700–714, 2022.

[9] M. Abebe, H. Lazu, and K. Daudjee. Tiresias: enabling predictive autonomous storage and indexing. *Proceedings of the VLDB Endowment*, 15(11):3126–3136, 2022.

[10] D. Agrawal and A. El Abbadi. The tree quorum protocol: An efficient approach for managing replicated data. In *VLDB*, volume 90, pages 243–254, 1990.

[11] N. Bronson, A. Aghayev, A. Charapko, and T. Zhu. Metastable failures in distributed systems. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 221–227, 2021.

[12] A. Charapko, A. Ailijiang, and M. Demirbas. Linearizable quorum reads in paxos. In *HotStorage*, 2019.

[13] R. Chirkova, A. Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *The VLDB Journal*, 11:216–237, 2002.

[14] B. Glasbergen, M. Abebe, and K. Daudjee. Tutorial: Adaptive replication and partitioning in data systems. In *Proceedings of the 19th International Middleware Conference Tutorials*, pages 1–5, 2018.

[15] B. Glasbergen, K. Langendoen, M. Abebe, and K. Daudjee. Chronocache: Predictive and adaptive mid-tier query result caching. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2391–2406, 2020.

[16] T. Harter, D. Borthakur, S. Dong, A. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Analysis of {HDFS} under hbase: a facebook messages case study. In *12th {USENIX} Conference on File and Storage Technologies ({FAST} 14)*, pages 199–212, 2014.

[17] 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*, pages 167–181, 2013.

[18] E. Kakoulli and H. Herodotou. Octopusfs: A distributed file system with tiered storage management. In *Proceedings of the 2017 acm international conference on management of data*, pages 65–78, 2017.

[19] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, et al. Scaling memcache at facebook. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 385–398, 2013.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[21] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database system concepts*. McGraw-Hill Education, 2011.

[22] A. J. Smith. Sequentiality and prefetching in database systems. *ACM Transactions on Database Systems (TODS)*, 3(3):223–247, 1978.

[23] R. Taft, N. El-Sayed, M. Serafini, Y. Lu, A. Aboulnaga, M. Stonebraker, R. Mayerhofer, and F. Andrade. P-store: An elastic database system with predictive provisioning. In *Proceedings of the 2018 International Conference on Management of Data*, pages 205–219, 2018.

[24] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems (TODS)*, 22(2):255–314, 1997.