Xylem: flexible and high-performance structured storage via dynamic data-flow

Jon Gjengset (MIT CSAIL)

Advisors: Malte Schwarzkopf, Eddie Kohler (Harvard), M. Frans Kaashoek, Robert Morris

Abstract. Xylem is a database design that targets read-heavy web applications. It materializes indexed query results to speed up reads, and automatically updates these cached results as writes arrive. Realized naively, this approach would incur significant memory overhead, especially when queries are similar, or when some results are rarely accessed. Xylem introduces two key techniques novel to streaming data-flow systems to overcome this: *partial materialization*, and *reuse* of shared operators and state. Using these techniques Xylem efficiently handles query set changes at runtime with minimal downtime, allowing applications to evolve over time. Experiments show that Xylem outperforms state-of-the-art systems, and adapts to new query sets while reads and writes remain live.

1. Research Problem and Motivation

An ideal storage system would provide the read performance of a key-value cache with the convenience and flexibility of relational queries. To emulate this ideal, web applications often supplement their database (DB) with fast in-memory caches (*e.g.*, memcached [20]). The durable, transactional DB offers a relational query interface to developers [5], and the cache efficiently serves pre-computed query results [23].

However, this two-tier construction comes at a price. First, the cache and the DB must now be coordinated: the application must invalidate entries in the cache on writes, and recompute query results on cache misses. This requires complex and error-prone techniques to ensure correct application semantics and avoid severe performance collapses [23]. Second, it lacks flexibility: adding new queries or changing the schema requires either discarding the cache and migrating the system via planned downtime, or implementing complex manual migrations [27]. Since web application queries and schemas can change multiple times per week [7, 9, 25], this poses a major headache for developers.

Xylem combines and extends ideas from databases and streaming data-flow systems to address these challenges. It transparently constructs and maintains a fast cache of pre-computed query results, supports relational queries, and allows online, backwards-compatible migrations to new schemas and queries. To provide these features, Xylem must overcome several intertwined problems: the space overhead of maintaining many cached query results can easily become prohibitive, existing techniques for updating cached results do not scale to many queries, and current stream processing systems cannot adapt to changing queries at runtime.

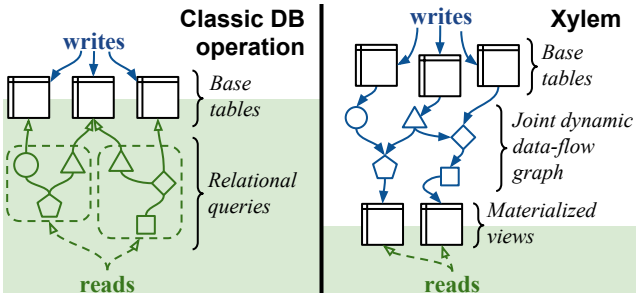

Figure 1: Classic databases compute read queries from base tables, while Xylem feeds writes through a dynamic, live-adapting data-flow graph to update materialize views.

2. Background and Related Work

Some DBs support *materialized views* of query results, and existing work shows how to choose [3, 12] and update [1, 16] them. Those that support incremental maintenance of such views are typically pull-based, *i.e.*, they update the views in response to data changes via whole-query re-evaluation, custom "triggers" [1, 14, 19], or incremental "delta queries" [2, 22]. Except in systems like SharedDB [11] that exploit view overlap, these views do not share state or compute, and if a new view is added, it must be computed afresh.

In contrast, *stream-processing* systems push updates through a fixed query plan to update materialized, windowed state [4, 6, 15, 26, 28]. *Data-flow* simplifies parallel update processing [13], including on streams [18, 21], and can incrementally update views that delta queries cannot [17]. However, with a fixed query plan, streaming and data-flow systems must discard all state and start anew to add queries.

Schema evolution systems automate database changes [8], but most operate offline (with the exception of F1 [24]).

3. Approach and Uniqueness

Figure 1 illustrates the central idea of Xylem. Instead of queries processing tuples from base-tables on every read, Xylem feeds new writes through a joint, dynamic data-flow graph to update materialized views of query results. Data-flow is attractive as it is amenable to multi-core and multi-machine distribution, and is inherently incremental. The key challenges to make this feasible are (*i*) to *minimize over-*

heads of materialized views and write-side updates, and (*ii*) to *quickly adapt* the data-flow graph as the query set changes. To achieve these, we developed two extensions to existing data-flow systems.

Query reuse. Queries often share common subexpressions, such as joins and aggregations. By materializing state for these expressions only once, we reduce overhead and amortize update cost. To detect common subexpressions, we apply an extended version of Finkelstein's query graphs [10]. When we add a new query, we reuse the query with the longest common subexpression prefix, and dynamically expand the data-flow graph from existing operators.

Partial materialization. Some queries see skewed key popularity distributions, and need not expend space and time on maintaining rarely-read results. Instead, Xylem can leave "holes" in its materializations for rare keys. Xylem can also support cache eviction by replacing existing records with holes. A read from such a key triggers a backwards query to fill in its result. Executing such backwards queries within an active data-flow system poses thorny consistency challenges: for example, Xylem must ensure that no write is reflected more than once in the result, even though updates for the same key may race in the data-flow graph.

Uniqueness. Xylem is unique in its ability to modify a data-flow computation without discarding existing state and while keeping reads and writes live. It also combines push-based streaming updates with pull-based backwards queries. State-of-the-art data-flow systems like Naiad [21] can do neither of these since they violate data-flow invariants (*e.g.*, past records cannot be re-sent) or implementation decisions (*e.g.*, static code generation for a fixed computation).

Cost. Xylem achieves fast reads by increasing memory usage and reducing consistency. Materialized views allow fast queries, but take up space proportional to the number of results they hold. Reduced consistency avoids internal locking and synchronization, but means that writes are not visible until they have propagated through the data-flow graph.

4. Results and Contributions

We implemented a prototype of Xylem in 20k lines of Rust. Below, we show that (*i*) Xylem outperforms both existing databases and the widely used MySQL/memcached stack, and (*ii*) that Xylem can migrate without downtime.

Figure 2 shows how throughput and latency scale with offered load in a simple news aggregator. 95% of requests read the title and vote count of a random article; the other 5% perform writes by voting for a random article. In MySQL, vote counts are manually materialized: an `UPDATE` statement increments the vote count on each write; the commercial "System Z" uses a materialized view. Both struggle to scale beyond 800k requests/sec. The two-tier MySQL/memcached stack scales to 2M requests/sec, but its read misses are costly due to the "thundering herd" problem [23, §3.2.1]. Xylem, by contrast, performs on-par with a memcached-only setup—

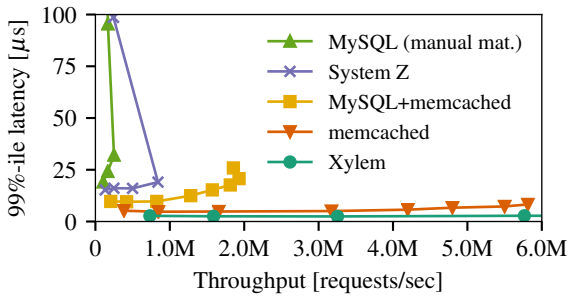

Figure 2: With 95% reads, Xylem outperforms MySQL, commercial System Z, and the MySQL/memcached stack; it performs on-par with (non-durable) memcached.

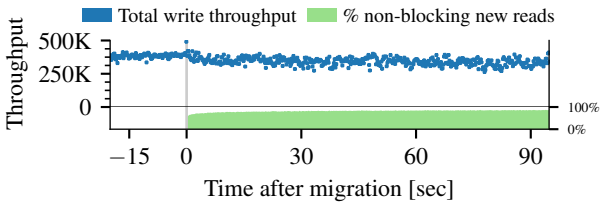

Figure 3: Xylem starts serving a new query without downtime when combining query reuse and partial materialization. Green is % of reads immediately satisfied (*i.e.*, no backwards query to fill in partial materialization).

even though memcached neither has durable storage nor a relational query interface. Both systems taper off beyond 5M requests/sec where they saturate all cores on the server, mostly doing read and write system calls. Other workload mixes, *e.g.*, with skewed popularity distributions or a 50% read/write workload, show similar results.

To test Xylem's live-migration, we change the application to use star ratings instead of binary "up/down" votes. Figure 3 shows that Xylem transitions without perceptible downtime. The old, vote-based view remains live throughout, allowing for a rolling application upgrade. The need to maintain both queries accounts for the drop in throughput from 400k to 330k writes/sec. Query reuse lets Xylem compute ratings using the existing vote counts rather than re-counting. Partial materialization builds the new materialized view incrementally as the system runs, and queries to fill "holes" are served using the reused vote counts, and have minimal impact on the throughput of new writes. Due to a skewed key popularity, over 90% of reads are served from the new materialized view after a few seconds.

Contributions. Xylem makes three primary contributions: (*i*) space- and time-efficient materialized view maintenance, *viz.* query reuse and partial materialization; (*ii*) live adaptation of data-flow computations while preserving existing state; and (*iii*) a prototype showing that Xylem outperforms competing setups and matches key-value store performance. Together, these allow us to improve web application performance while reducing complexity for developers.

References

[1] Parag Agrawal, Adam Silberstein, Brian F. Cooper, Utkarsh Srivastava, and Raghu Ramakrishnan. "Asynchronous View Maintenance for VLSD Databases". In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. Providence, Rhode Island, USA, June 2009, pp. 179–192.

[2] Yanif Ahmad, Oliver Kennedy, Christoph Koch, and Milos Nikolic. "DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views". In: *Proceedings of the VLDB Endowment* 5.10 (June 2012), pp. 968–979.

[3] Khalil Amiri, Sanghyun Park, Renu Tewari, and Sriram Padmanabhan. "DBProxy: a dynamic data cache for web applications". In: *Proceedings of the 19th International Conference on Data Engineering (ICDE)*. Mar. 2003, pp. 821–831.

[4] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, et al. *STREAM: The Stanford Data Stream Management System*. Technical Report 2004-20. Stanford InfoLab, 2004.

[5] David F. Bacon, Nathan Bales, Nico Bruno, Brian F. Cooper, Adam Dickinson, Andrew Fikes, Campbell Fraser, et al. "Spanner: Becoming a SQL System". In: *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. Chicago, Illinois, USA, 2017, pp. 331–343.

[6] Badrish Chandramouli, Jonathan Goldstein, Mike Barnett, Robert DeLine, Danyel Fisher, John C. Platt, James F. Terwilliger, et al. "Trill: A High-performance Incremental Query Processor for Diverse Analytics". In: *Proceedings of the VLDB Endowment* 8.4 (Dec. 2014), pp. 401–412.

[7] Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. "Automating the Database Schema Evolution Process". In: *The VLDB Journal* 22.1 (Feb. 2013), pp. 73–98.

[8] Carlo A. Curino, Hyun J. Moon, and Carlo Zaniolo. "Graceful Database Schema Evolution: The PRISM Workbench". In: *Proceedings of the VLDB Endowment* 1.1 (Aug. 2008), pp. 761–772.

[9] Dror G. Feitelson, Eitan Frachtenberg, and Kent L. Beck. "Development and Deployment at Facebook". In: *IEEE Internet Computing* 17.4 (July 2013), pp. 8–17.

[10] Sheldon Finkelstein. "Common Expression Analysis in Database Applications". In: *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data*. Orlando, Florida, USA, June 1982, pp. 235–245.

[11] Georgios Giannikis, Gustavo Alonso, and Donald Kossmann. "SharedDB: Killing One Thousand Queries with One Stone". In: *Proceedings of the VLDB Endowment* 5.6 (Feb. 2012), pp. 526–537.

[12] Himanshu Gupta and Inderpal Singh Mumick. "Selection of views to materialize in a data warehouse". In: *IEEE Transactions on Knowledge and Data Engineering* 17.1 (Jan. 2005), pp. 24–43.

[13] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. "Dryad: Distributed Data-parallel Programs from Sequential Building Blocks". In: *Proceedings of the 2nd ACM SIGOPS European Conference on Computer Systems (EuroSys)*. Lisbon, Portugal, Mar. 2007, pp. 59–72.

[14] Bryan Kate, Eddie Kohler, Michael S. Kester, Neha Narula, Yandong Mao, and Robert Morris. "Easy Freshness with Pequod Cache Joins". In: *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Seattle, Washington, USA, Apr. 2014, pp. 415–428.

[15] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, et al. "Twitter Heron: Stream Processing at Scale". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. Melbourne, Victoria, Australia, May 2015, pp. 239–250.

[16] Ki Yong Lee and Myoung Ho Kim. "Optimizing the Incremental Maintenance of Multiple Join Views". In: *Proceedings of the 8th ACM International Workshop on Data Warehousing and OLAP (DOLAP)*. Bremen, Germany, Nov. 2005, pp. 107–113.

[17] Frank McSherry. *Implementing a TPC-H-like evaluation*. Blog post. URL: https://github.com/frankmcsherry/blog/blob/master/posts/2017-04-24.md (visited on 05/01/2017).

[18] Frank McSherry, Derek G. Murray, Rebecca Isaacs, and Michael Isard. "Differential dataflow". In: *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR)*. Asilomar, California, USA, Jan. 2013.

[19] John Meehan, Nesime Tatbul, Stan Zdonik, Cansu Aslantas, Ugur Cetintemel, Jiang Du, Tim Kraska, et al. "S-Store: Streaming Meets Transaction Processing". In: *Proceedings of the VLDB Endowment* 8.13 (Sept. 2015), pp. 2134–2145.

[20] *memcached - a distributed memory object caching system*. URL: https://www.memcached.org/ (visited on 09/15/2017).

[21] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. "Naiad: A Timely Dataflow System". In: *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP)*. Farmington, Pennsylvania, USA, Nov. 2013, pp. 439–455.

[22] Milos Nikolic, Mohammad Dashti, and Christoph Koch. “How to Win a Hot Dog Eating Contest: Distributed Incremental View Maintenance with Batch Updates”. In: *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. San Francisco, California, USA, 2016, pp. 511–526.

[23] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, et al. “Scaling Memcache at Facebook”. In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. Lombard, Illinois, USA, Apr. 2013, pp. 385–398.

[24] Ian Rae, Eric Rollins, Jeff Shute, Sukhdeep Sodhi, and Radek Vingralek. “Online, Asynchronous Schema Change in F1”. In: *Proceedings of the VLDB Endowment* 6.11 (Aug. 2013), pp. 1045–1056.

[25] Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. “Continuous Deployment at Facebook and OANDA”. In: *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. Austin, Texas, USA, 2016, pp. 21–30.

[26] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, et al. “Storm@Twitter”. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. Snowbird, Utah, USA, June 2014, pp. 147–156.

[27] Jacqueline Xu. *Online migrations at scale*. Stripe engineering blog. URL: https://stripe.com/blog/online-migrations (visited on 02/01/2017).

[28] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. “Discretized Streams: Fault-tolerant Streaming Computation at Scale”. In: *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP)*. Farmington, Pennsylvania, USA, Nov. 2013, pp. 423–438.