Materialized Views and Key-Value Pairs in a Cache Augmented SQL System: Similarities and Differences

Shahram Ghandeharizadeh, Jason Yap

Database Laboratory Technical Report 2012-05

Computer Science Department, USC

Los Angeles, California 90089-0781

September 10, 2012

Abstract

In the era of no "one-size-fits-all", organizations extend a relational database management system (RDBMS) with a key-value store (KVS) to enhance the velocity of big data applications with a high read to write ratio. A popular in-memory KVS is memcached in use by well known Internet destinations such as YouTube and Wikipedia. Its simple interface provides put, get, and delete of key-value pairs computed using tabular data. A key question is how do these key-value pairs compare with materialized views in a RDBMS. This paper provides an answer to this question.

A Introduction

What is a view?

A view is a virtual table defined using an expression that references other tables in a relational database management system (RDBMS). It is re-computed every time a query references the view. A view might be authored using SQL, relational algebra [13], datalog and others [6]. This paper assumes SQL.

What is a materialized view?

A materialized view (MV) stores the tuples of the view in the database. One may construct index structures on the materialized view. Hence, accesses to the view are much faster than re-computing it. Typically, a database administrator (DBA) analyzes the workload of an application to authors MVs and their indicies. For an example with data warehousing queries see [24]. This study shows MVs enhance the performance of row-stores significantly. Selecting which virtual views to materialize, the *view selection problem*, has been studied extensively [21, 13, 14].

It is time consuming for a RDBMS to materialize a view and its indices. Hence, in the presence of updates to the base tables referenced by a MV, it is not efficient to drop the MV. Instead, MVs are maintained up to date *incrementally* [13, 20, 18]. This approach computes changes to the MV and applies them to the MV to bring it up to date.

A query optimizer may employ a MV to process SQL queries that do not reference it explicitly [22, 18]. Moreover, a physical database design adviser may recommend index structures on a MV as it is a table [1, 5].

What is a Key-Value Store (KVS)?

A KVS maintains key-value pairs consisting of a unique identifier (key) associated with some arbitrary data (value). It provides a simple interface such as put, get, and delete to store, retrieve, and delete key-value pairs. It provides little or no ability to interpret its value with no query mechanism for the content of the values [23, 7]. A popular KVS is memcached in use by many popular Internet destinations such as YouTube and Wikipedia.

What is a Cache Augmented SQL (CASQL) system?

Cache Augmented SQL, CASQL, systems are an important class of distributed systems, targeting applications with a high read to write ratios. These systems augment a RDBMS with a KVS to enhance overall velocity of operations that retrieve and process a very small amount of the entire data set [16, 8, 25, 10, 9, 17, 2, 3, 19, 15, 11]. They may materialize either the results of a query (key=query string, value=result set computed by the RDBMS) or a code segment (key=unique identifier for the code segment constructed using its input, value=output of the code segment) as key-value pairs in the KVS. This enhances performance because a cache look up is much faster than executing either a SQL query or a code segment that issues several queries. In the presence of updates to the tabular data, a CASQL solution may maintain the cached key-value pairs consistent transparently [8, 9, 10, 2, 3, 19, 15, 11].

How are materialized views similar to cached key-value pairs?

Both MV and key-value pairs store a separate physical copy of the tabular data. This copy must be maintained consistent with the base tables. The RDBMS automatically maintains MVs consistent and serialize transactions to provide ACID properties, e.g., by using the REFRESH ON COMMIT. Similarly, transparent caching techniques maintain key-value pairs consistent in the presence of updates to the RDBMS. For example, TxCache [19] implements snap-shot isolation and one may configure SQLTrig [11] to implement serial schedules.

Both MVs and key-value pairs might be used to enhance velocity of data retrieval by approximating the final answers of a posed query. For example, an application may utilize "REFRESH ON DEMAND" option when authoring a MV and update it periodically. Queries processed using such views may observe stale

data. Similarly, a CASQL system may incrementally update key-value pairs and cause the application to observe either stale data [17, 11] or suffer from dirty reads [15].

How are materialized views different than cached key-value pairs?

MVs and key-value pairs are suitable for different application classes. MVs enhance performance of decision support applications and their On-Line Analytical Processing (OLAP). KVS and key-value pairs enhance performance of queries that read a very small amount of entire dataset repeatedly. Thus, one is not a substitute for the other. We elaborate on this below.

SQL queries used to compute a MV typically retrieve many rows. It is not uncommon to find index structures on a MV to expedite processing of SQL queries that reference it. In contrast, a key-value pair corresponds to an SQL query (or a code segment) that is very selective (outputs a few values), e.g., retrieve the profile information of a member of a social networking site given the user's login and password. A CASQL enhances the performance of interactive operations when its key-value pairs are accessed far more frequently than they are updated. This is because a key-value look up is faster than processing SQL queries [12].

With a CASQL, there may exist millions (if not billions) of key-value pairs pertaining to different instances of a simple SQL query whose results are cached as key-value pairs in the KVS. For example, with a social networking application, each SQL query issued on behalf of a member to retrieve her profile might be a key-value pair in the KVS. In contrast, there exists a few (in the order of tens of) MVs authored by a database designer to enhance overall system performance based on a known or expected workload.

SQL queries that are the basis of a key-value pair with a CASQL are much faster to execute than those that are the basis of a MV with a RDBMS. This explains why RDBMSs *maintain* MVs instead by incrementally updating them while CASQL systems *invalidate* key-value pairs by deleting and re-computing them.

Finally, a MV is typically created by a human and crafted to specifically meet the expected needs for an OLAP workload. The workload should be known in advance in order to use MVs effectively. On the other hand, a CASQL with a transparent cache generates key-value pairs dynamically as a workload executes. It does not require advanced knowledge of the workload. (When a CASQL in employed non-transparently, a human participates to identify code segments whose results should be cached.)

Can MVs and key-value pairs co-exist?

MVs and key-value pairs are implemented by different components and may co-exist. While a RDBMS implements MVs, a KVS implements key-value pairs. Thus, one may use the RDBMS of a CASQL to author MVs to enhance processing of OLAP queries. And use its KVS to cache the result of OLAP queries in order to expedite the processing of those issued repeatedly. A query optimizer extended with a cache manager for data warehouses and data marts was explored in [22]. Extensions of these ideas to a CASQL is

a future research direction.

Is it possible to use MVs as a substitute for key-value pairs?

MVs are not a substitute for key-value pairs. We show this using the view profile action of the BG benchmark [4]. This action is a simple SQL query that retrieves profile information (a row) of a Member table with 10,000 rows (members). BG issues 100,000 view profile requests in turn. Each request references a member. Using a commercial RDBMS, the average execution time of this query is 2.5 milliseconds. If one defines 10,000 MVs (one for each query) and executes the same workload, the average execution time of each query increases to 6 milliseconds. Using a CASQL with a cold cache, the average execution of the query is 1.5 milliseconds. A warm cache with 10,000 key-value pairs (one per query) reduces this time to 0.3 milliseconds.

References

- [1] S. Agrawal, N. Bruno, S. Chaudhuri, and V. Narasayya. AutoAdmin: Self-Tuning Database Systems Technology. *IEEE Data Engineering Bulletin*, 29:7–15, 2006.
- [2] C. Amza, A. L. Cox, and W. Zwaenepoel. A comparative evaluation of transparent scaling techniques for dynamic content servers. In *ICDE*, 2005.
- [3] C. Amza, G. Soundararajan, and E. Cecchet. Transparent Caching with Strong Consistency in Dynamic Content Web Sites. In *Supercomputing*, ICS '05, pages 264–273, New York, NY, USA, 2005. ACM.
- [4] S. Barahmand and S. Ghandeharizadeh. BG: A Benchmark to Evaluate Interactive Social Networking Actions. In *Submitted for Publication*, 2012.
- [5] N. Bruno and S. Chaudhuri. Physical Design Refinement: The "Merge-Reduce" Approach. In *In International Conference on Extending Database Technology (EDBT*, 2006.
- [6] N. Bruno and S. Chaudhuri. Constrained Physical Design Tuning. In *In PVLDB*, 2008.
- [7] R. Cattell. Scalable SQL and NoSQL Data Stores. SIGMOD Rec., 39:12–27, May 2011.
- [8] J. Challenger, P. Dantzig, and A. Iyengar. A Scalable System for Consistently Caching Dynamic Web Data. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, 1999.
- [9] A. Datta, K. Dutta, H. Thomas, D. VanderMeer, D. VanderMeer, K. Ramamritham, and D. Fishman. A Comparative Study of Alternative Middle Tier Caching Solutions to Support Dynamic Web Content Acceleration. In *VLDB*, pages 667–670, 2001.

- [10] L. Degenaro, A. Iyengar, I. Lipkind, and I. Rouvellou. A Middleware System Which Intelligently Caches Query Results. In *IFIP/ACM International Conference on Distributed systems platforms*, 2000.
- [11] S. Ghandeharizadeh and J. Yap. SQL Query To Trigger Translation: A Novel Consistency Technique for Cache Augmented DBMSs. In *Submitted for Publication*, 2012.
- [12] S. Ghandeharizadeh, J. Yap, and S. Barahmand. COSAR-CQN: An Application Transparent Approach to Cache Consistency.
- [13] A. Gupta and I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.
- [14] H. Gupta. Selection of Views to Materialize in a Data Warehouse. In *Proceedings of ICDT, Lecture Notes in Computer Science, 1186 Springer 1997*, January 1997.
- [15] P. Gupta, N. Zeldovich, and S. Madden. A Trigger-Based Middleware Cache for ORMs. In *Middle-ware*, 2011.
- [16] A. Iyengar and J. Challenger. Improving Web Server Performance by Caching Dynamic Data. In *In Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 49–60, 1997.
- [17] A. Labrinidis and N. Roussopoulos. Exploring the Tradeoff Between Performance and Data Freshness in Database-Driven Web Servers. *The VLDB Journal*, 2004.
- [18] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. Materialize View Selection and Maintenance Using Multi-Query Optimization. In *Proceedings of ACM SIGMOD*, May 2001.
- [19] D. R. K. Ports, A. T. Clements, I. Zhang, S. Madden, and B. Liskov. Transactional consistency and automatic management in an application data cache. In *OSDI*. USENIX, October 2010.
- [20] K. Ross, D. Srivastava, and S. Sudarshan. Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. In *Proceedings of ACM SIGMOD*, May 1996.
- [21] N. Roussopoulos. View indexing in relational databases. *ACM Trans. Database Syst.*, 7(2):258–290, June 1982.
- [22] P. Roy, K. Ramamritham, S. Seshadri, P. Shenoy, and S. Sudarshan. Don't Trash your Intermediate Results, Cache 'em. *CoRR*, cs.DB/0003005, 2000.
- [23] M. Stonebraker and R. Cattell. 10 Rules for Scalable Performance in Simple Operation Datastores. *Communications of the ACM*, 54, June 2011.

- [24] V. Valloppilli and K. Ross. Cache Array Routing Protocol, V 1.0, Internet Draft, http://icp.ircache.net/carp.txt, Feb. 1998.
- [25] K. Yagoub, D. Florescu, V. Issarny, and P. Valduriez. Caching Strategies for Data-Intensive Web Sites. In *VLDB*, pages 188–199, 2000.