VLDB 2009
35th International Conference on Very Large Data Bases
August 24-28, 2009
Lyon
France

Reviews For Paper

Track	Core Database Technology
Paper ID	660
Title	Transactional Caching of Application Data Using Recent Snapshots

---

Masked Reviewer ID: Reviewer 1

Review:

Question	
Overall Rating	Neutral
Reject due to technical incorrectness	No
Novelty	Medium
Technical Depth	Medium
Presentation	Adequate
Summary of the paper's main contributions and impact (up to one paragraph)	By using a multi-versioning technique, the proposed caching scheme enables reading a consistent snapshot of cached values at a specified (past) point of time. Each cached value is versioned and tagged with its validity interval. On a query execution, the right set of versions are chosen by comparing the transaction's timestamp with the tagged validity intervals.
Three strong points of the paper (please number them S1,S2,S3)	S1. Implementation-level complex issues such as avoiding snapshot isolation anomalies (Section 3.2), lazy timestamp selection (Section 6.2) are explored in detail. S2. Good arguments on the correctness of the proposed complex algorithms.
Three weak points of the paper (please number them W1,W2,W3)	W1. Exploiting user-specified freshness constraint and the multi-versioning technique has been already explored in the context of database replication. Since caching is conceptually similar with replication, the proposed technique can be seen as an extension of the already-existing solutions. Or, the authors have to more clearly differentiate from them. W2. This paper simplified the addressed problem by caching only key-value pairs in a hash table. There is no query completeness check issue as in IBM DBCache. And also, this paper assumes a 'single shared cache' (in Section 8) which partitions the hash table to multiple cache nodes. There is no cache coherence management issues as in Oracle RAC. W3. The experiment need to be more extensive as pointed in my comments #2 ~ #5.

Detailed Comments (please number each point)	1.The proposed cache here lies between applications and end-users, not between database and applications. This has trade-offs. It may reduce end-user latency on cache hit, but the relationship between cacheable functions and the database state must be carefully checked (as the authors mentioned in Section 2.1). *Regarding the experiments in this paper: 2. The tested database size was 1.1GB while 1GB of memory (512MB * 2) is allocated for TxCache. This will not be a realistic configuration in general cases. 3. As the freshness requirement increases, the throughput increases as in Figure 5. But, higher freshness requirement will consume more memory. The actual memory consumption of TxCache and 'no-caching' need to be compared. 4. In the experiment, this paper compared TxCache only with 'no-caching' case. It seems necessary to compare with other caches such as memcached or other transactional caches. 5. TxCache was tested only with two app/cache servers. This seems not enough to show scalability of TxCache.

Masked Reviewer ID: Reviewer 2

Review:

Question	
Overall Rating	Strong Accept
Reject due to technical incorrectness	No
Novelty	High
Technical Depth	High
Presentation	Very Good
Summary of the paper's main contributions and impact (up to one paragraph)	The paper presents a query cache that guarantees that transactions read consistent data even if some queries are read from the cache and others from the database. Queries see a committed snapshot of the data as either start of transaction or some recent time in the past. The database engine is extended to provide sufficient information to the cache to decide which entry a particular transactin may read and also to provide snapshots in the past. This paper presents a very sophisticated solution that takes advantage of the internals of the PostgeSQL version system, and handles complex issues such as phantoms.

Three strong points of the paper (please number them S1,S2,S3)	S1: Sophisticated, smart solution S2: thorough solution, considering query predicates, etc. S3: complete implementation with the PostgreSQL engine
Three weak points of the paper (please number them W1,W2,W3)	W1: As always, the performance evaluation could be more extensive W2: some related work is missing
Detailed Comments (please number each point)	1. It would have been nice to see what happens on larger data sets and when the db does not fit into main memory. 2. Freshness has also been well studied in replicated systems. Plattner and Alonso's middleware system allows to run queries in the past on the satellite databases. Roehm et. al. (VLDB 2002) refresh secondary database replicas only when necessary. Bernstein et. al in Sigmod 2006 discuss relaxed currency for middle-tier caching. It would have been necessary how their approach differs conceptually from those approaches. 3. The paper indicates that the invalidity mask is the union of the validity times for all tup;les that failed the visibility check. This can be easily done if the system scans through all the existing versions of all tuples. But there might be execution plans that don't do this, e.g., index accesses. How would these tuples or the invalidity mask then be constructed?
List specific clarifications you seek from the Authors (if you have answered "Yes" to Q. 6) Use this space to respond to author feedback too.	REPLACE THIS WITH YOUR ANSWER

Masked Reviewer ID: Reviewer 3

Review:

Question	
Overall Rating	Reject
Reject due to technical incorrectness	No
Novelty	Low
Technical Depth	Medium
Presentation	Adequate

Summary of the paper's main contributions and impact (up to one paragraph)	The paper proposes to allow queries to run against slightly stale snapshots (using versions kept anyway in multiversion platforms like postgres)
Three strong points of the paper (please number them S1,S2,S3)	S1. There are some technical engineering contributions in tracking validity and dealing with phantoms S2. There is a well-written argument for correctness
Three weak points of the paper (please number them W1,W2,W3)	W1. There isn't a clear account of how this improves on previous work which also have the idea of using slightly stale copies for improving performnce of read-only activities see eg Plattner VLDBJ in 2008, following conferences Middleware 04 and 06 and demo at sigmod'06, which itself built on Roehm in vldb'02 (the main difference I see is that this new paper does its work in the application cache in memory). We need to see an argument about whether the difference in system architecture really requires a new set of solution techniques, or whether by using Plattner's algorithmic ideas in an application-level object cache, one would have just as good a technique as this new proposal. w2. The evaluation is only against doing no caching at all, or showing the performance impact of changing the staleness limit [without showing the counter-effect on semantics] There is no comparison to the performance of the existing ideas from Plattner (when they are ported to this paper's setting of application caches)
Detailed Comments (please number each point)	1. The authors should reference the many papers in our community that already follow the idea of using slightly stale data for read-only activities. It is the authors' responsibility to indicate how their work differs from this, and in the evaluation to compare their work to these existing schemes. Important work includes Roehm (VLDB'02), Plattner (VLDBJ in 2008) and Bernstein (sigmod'06). The latter has a validity interval concept (including techniques for intersecting intervals, and for approximating them) that should be compared to the one proposed here. Note that comparison means eg comparing performance to what one gets by using the ideas from Plattner in the application-cache setting; it is not enough just to say that the previous papers are in a different system architecture.