Aspects of Append-Based Database Storage Management on Flash Memories

Robert Gottstein
Databases and Distributed Systems Group
TU-Darmstadt Germany
gottstein@dvs.tu-darmstadt.de

Ilia Petrov
Data Management Lab
Reutlingen University, Germany
ilia.petrov@reutlingen-university.de

Alejandro Buchmann
Databases and Distributed Systems Group
TU-Darmstadt, Germany
buchmann@dvs.tu-darmstadt.de

***Abstract*—New storage technologies, such as Flash and Non-Volatile Memories, with fundamentally different properties are appearing. Leveraging their performance and endurance requires a redesign of existing architecture and algorithms in modern high performance databases. Multi-Version Concurrency Control (MVCC) approaches in database systems, maintain multiple timestamped versions of a tuple. Once a transaction reads a tuple the database system tracks and returns the respective version eliminating lock-requests. Hence under MVCC reads are never blocked, which leverages well the excellent read performance (high throughput, low latency) of new storage technologies. Upon tuple updates, however, established implementations of MVCC approaches (such as Snapshot Isolation) lead to multiple random writes – caused by (i) creation of the new and (ii) in-place invalidation of the old version – thus generating suboptimal access patterns for the new storage media. The combination of an append based storage manager operating with tuple granularity and snapshot isolation addresses asymmetry and in-place updates. In this paper, we highlight novel aspects of log-based storage, in multi-version database systems on new storage media. We claim that multi-versioning and append-based storage can be used to effectively address asymmetry and endurance. We identify multi-versioning as the approach to address data-placement in complex memory hierarchies. We focus on: *version handling*, (physical) *version placement*, *compression* and *collocation* of tuple versions on Flash storage and in complex memory hierarchies. We identify possible read- and cache-related optimizations.**

Keywords-Multi Version Concurrency Control; Snapshot Isolation; Version; Append Storage; Flash; Data Placement.

I. INTRODUCTION

New storage technologies such as flash and non-volatile memories have fundamentally different characteristics compared to traditional storage such as magnetic discs. Performance and endurance of these new storage technologies highly depend on the I/O access patterns.

Multi-Version approaches maintaining versions of tuples, effectively leverage some of their properties such as fast reads and low latency. Yet, asymmetry and slow in-place updates need to be addressed on architectural and algorithmic levels of the DBMS. Snapshot Isolation (SI) has been implemented in many commercial and open-source systems: Oracle, IBM DB2, PostgreSQL, Microsoft SQL Server 2005, Berkeley DB, Ingres, etc. In some systems, SI is a separate isolation level, in others used to handle serializable isolation.

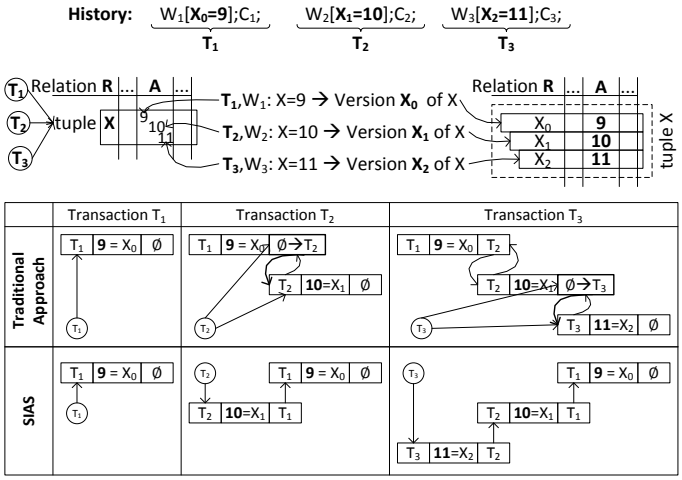

Figure 1. Invalidation in SI and SIAS

Under the concept of *Append-based storage* management any newly written data appended is at the logical head of a circular append log. This way, random writes are transformed into sequential writes and in-place update operations are reduced to a controlled append of the data, which is an effective mechanism to address the assymmetric performance of new storage technologies (see Section III)

In SIAS [1], we combine snapshot isolation and append storage management (with tuple granularity) on Flash. Under TPC-C workload SIAS achieves up to 4x performance improvement on Flash SSDs, a significant *write overhead reduction* (up to 38x), better *space utilization* due to denser version packing per page, better *I/O parallelism* and up to 4x lower disk I/O execution times, compared to traditional approaches. SIAS aids better *endurance*, due to the use of out-of-place writes as appends and write overhead reduction.

SIAS implicitly invalidates tuple versions by creating a successor version; thus, avoiding in-place updates. SIAS manages tuple versions of a single data item as simply linked lists (chains), addressed by a virtual tuple ID (VID). Figure 1 illustrates the invalidation process in SI and SIAS. Transactions $T1$, $T2$, $T3$ update data item X in serial order. Thereafter, the relation contains three different tuple versions of data item X. The initial version X_0 of X is created by $T1$ and updated by $T2$. The *traditional approach (SI)* invalidates X_0 in-place by physically setting the invalidation timestamp

 ISBN: 978-1-61208-247-9

and creating X_1. Analogously, $T3$ updates X_1 with the physical in-place invalidation of X_1. *SIAS* connects tuple versions using the VID where the newest tuple version is always known. Each tuple maintains a backward reference to its predecessor, which does not need to be updated in place. Hence, updating X_0 leads to the creation of X_1.

We report our work in progress on data placement and summarize key findings and the preliminary results of SIAS (published in a previous work). In this paper, we focus on novel aspects of *version handling*, (physical) *placement* and *collocation* on append-based database storage manager using flash memory as primary storage.

In the next section we present the related work. Section III provides a brief summary of the properties of flash technology. Section IV introduces the SIAS approach, aspects of *version handling*, (physical) *placement* and *collocation*. Section V concludes the paper.

II. Related Work

SIAS organizes data item versions in simple chronologically ordered chains, which has been proposed by Chan et al. in [2] and explored by Petrov et al. in [3] and Bober et al. in [4] in combination with MVCC algorithms and special locking approaches. Petrov et al. [3], Bober et al. [4], Chan et al. [2] explore a log/append-based storage manager. The applicability of append-based database storage management approaches for novel asymmetric storage technologies has been partially addressed by Stoica et al. in [5] and Bernstein et al. in [6] using page-granularity, whereas SIAS employs tuple-granularity much like the approach proposed by Bober et al. in [4], which, however, invalidates tuples in-place. Given a page granularity the whole invalidated page is remapped and persisted at the head of the log, hence no write-overhead reduction. In tuple-granularity, multiple new tuple-versions can be packed on a new page and written together. Log storage approaches at file system level for hard disk drives have been proposed by Rosenblum in [7]. A performance comparison between different MVCC algorithms is presented by Carey et al. in [8]. Insights to the implementation details of SI in Oracle and PostgreSQL are offered by Majumdar in [9]. An alternative approach utilizing transaction-based tuple collocation has been proposed by Gottstein et al. in [10]. Similar chronological-chain version organization has been proposed in the context of update intensive analytics by Gottstein et al. in [11]. In such systems data-item versions are never deleted, instead they are propagated to other levels of the memory hierarchy such as HDDs or Flash SSDs and archived. Any logical modification operation is physically realized as an append. SIAS on the other hand provides mechanisms to couple version visibility to (logical and physical) space management. SIAS uses transactional time (all timestamps are based on a transactional counter) in contrast to timestamps that correlate to logical time (dimension). Stonebraker et al. realized the concept of TimeTravel in PostgreSQL [12].

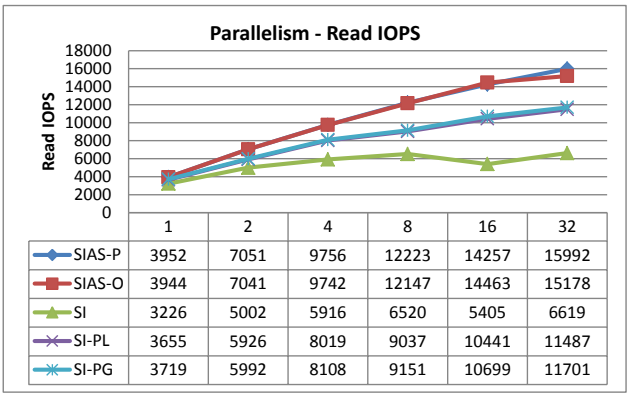

Figure 2. I/O Parallelism on Intel X25-E SSD - 60 Minute TPC-C

III. Flash Memories

We briefly sum up common properties of flash storage and compare them to traditional storage technologies (RAM, spinning disks): (i) *read/write asymmetry* – reads are much faster than writes, up to an order of magnitude; (ii) *low random write throughput* – small random writes are 5-10x slower than reads; (iii) *endurance issues and wear*; (iv) *suboptimal mixed load performance*: mixing reads/writes or random/sequential patterns leads to performance degradation.

Table I
SIAS and SI results on Intel X25-E SSD [1]

Queue Depth 1

Trace	read IOPS	write IOPS	read MB	write MB	time (sec)
SIAS-O(I)	4476	20	19713	89.96	563.675
SIAS-P (I)	4499	19	20666	89.96	587.873
SI (I)	3771	322	19901	1624	721.843
SIAS-O(II)	3947	13	11542	39.76	374.204
SIAS-P (II)	3953	13	11562	39.76	374.341
SI (II)	3656	432	11852	1395	414.869

Queue Depth 32

Trace	read I/O	write I/O	read MB	write MB	time (sec)
SIAS-O(I)	14500	66	19713	89.96	174.01
SIAS-P (I)	14642	63	20666	89.96	180.658
SI (I)	3360	264	19901	1624.9	805.193
SIAS-O(II)	15981	55	11542	39.76	92.44
SIAS-P (II)	15722	54	11562	39.76	94.128
SI (II)	11365	1338	11852	1395	133.478

IV. SIAS - Snapshot Isolation Append Storage

In this section we provide a short summary of the SIAS approach [1]. SIAS manages versions as simply linked lists (chains) that are addressed by using a virtual tuple ID (VID). On creation of a new version it implicitly invalidates the old one resulting in an out-of-place write – implemented as a logical append – and avoiding the in-place update of the predecessor. SIAS is coupled to an append-based storage manager, appending in units of tuple versions. Table I shows our test results with SIAS. Two traces containing all accessed and inserted tuples were recorded under PostgreSQL running TPC-C instrumented with different

 ISBN: 978-1-61208-247-9

parameters. *Trace I* was instrumented using 5 warehouses with four hours runtime and *Trace II* using 200 warehouses and 90 minutes runtime. Both traces were fed into our database storage simulator which generated SIAS-O/P and SI traces, containing read and written DB-pages to be used as input for the FIO benchmark which executed them on an Intel X25-E SSD. SIAS-O is a simulation with and SIAS-P without caching of the SIAS data structures, where SI is the classic Snapshot Isolation using in-place updates on the invalidation. The conclusions of our results are: *(i) SI reads more than SIAS-O but less than SIAS-P; (ii) SI writes more gross-data than SIAS-O/P; (iii) SIAS-O/P reads with more IOPS than SI; (iv) SIAS needs less runtime than SI; and (v) SIAS-O/P scales better than SI with higher parallelism.* We also conducted tests using SI and page-wise append, performing a remapping of all pages which either appends pages local at each relation (SI-PL) or at a global append area (SI-PG) with the results displayed in Figure 2. We found that while each of both outperforms original SI by 15 to 76%, both themselfes are outperformed by SIAS-O/P by 6 to 36%. Our results empirically confirm our hypothesis that (a) appends are more suitable for Flash, (b) append granularity is crucial to performance and (c) appending in tuples and writing in pages is superior to remapping of pages. In the following sections we describe our approaches to merging of pages and physical tuple version placement as well as compression and indexing.

A. Merge

One key assumption of append based storage is that once data was appended it is never updated in-place. In a multi-version database old and updated versions inevitably become invisible which leads to different tuple versions of the same data item, most likely located at different physical pages. Hence pages *age* during runtime and contain visible and invisible tuple versions. In a production database running 24x7 it is realistic to assume that *net amount of visible tuples* on such pages is low and that an ample amount of outdated *dead* tuple versions is transferred, causing cache pollution. Once a certain threshold of dead tuples per page is reached it is beneficial to re-insert still visible tuples and mark the page as invalid. Dead tuples may be pruned or archived. Since a physical invalidation of the old page would lead to an in-place update, we suggest using a bitmap index providing a boolean value per page indicating its invalidation. The page address correlates to the position in the bitmap index, therefore the size is reasonably small. A merge therefore includes the re-insertion of still visible tuples into a new page and the update of the bitmap index. On the re-insertion the placement of the tuples may be reconsidered (Sect. IV-B).

Space reclamation of invalidated pages is also known as garbage collection in most MVCC approaches. On flash memories, a physical erase can only be executed in erase unit granularities, hence it makes sense to apply reclamation in such granules and to make use of the *Trim* command. Pruning a single DB-page with the size smaller than an erase unit will most likely cause the FTL to create a remapping within the it's logical/physical block address table and postpones the physical erasure. This may result in unpredictable latency outliers due to fragmentation and postponed erasures [3]. Using the bitmap index, indicating deleted/merged pages (prunable), a consecutive sequence of pruned pages within an erase unit can be selected as a victim altogether. If the sequence still contains pages which have not been merged yet, they can be merged before the reclamation.

SIAS uses data structures to guarantee the access to the most recent committed version X_v of a data item X. If only the most recent committed version has to be re-inserted (i.e. no successor version exists), nothing but the SIAS data structure has to be updated. It is theoretically possible that the tuple version is still visible and invalidated. In this case a valid successor version to that tuple exists which has to be re-inserted as well: Let P_m be the victim page, X_i an invalidated tuple version of data item X, where $X_i \in P_m$ and $X_v \in P_k$, $P_m \neq P_k$. X_v is the direct successor to X_i physically pointing to X_i. The merge of P_m leads to a re-insertion of X_i as X_i* which leads to a re-insertion of X_v as X_v*, pointing to X_i*. The SIAS data structures are updated such that the most recent committed version of X know is X_i*. It is not necessary to merge P_k as well, since X_v simply becomes an orphan tuple version which is not reachable by the SIAS data structures. Phantoms cannot occur since X_v* and X_v yield the same VID and version count. Nevertheless, it is most likely that X_i will become invisible during the merge since OLTP transactions are usually short and fast running.

B. Tuple Version Placement

In SIAS, each relation maintains a private append region and tuples are appended in the order they arrive at the append storage manager. Tuples of different relations are not stored into the same page and pages of different relations are not stored into the same relation regions. Appending tuple versions in the order they arrive may be suboptimal, since merged, updated and inserted tuples usually have different access frequencies. Collocation of tuples according to their access frequency can be benefitial since the net amount of actually used tuples per transferred page is higher [10]. Using temperature as a metric, often accessed tuples are hot and seldom accessed tuples are cold. The goal of tuple placement is to transfer as much hot tuples as possible with one I/O to reduce latency and to group cold tuples such that archiving and merging is efficiently backed. Visibility meta-information also contributes to access frequency, since tuples need to be checked for visibility. This creates yet another dimension upon which tuples can be related apart from the attribute values. Even if the content is not related the visibility of the tuples may be comparable.

 ISBN: 978-1-61208-247-9

Under the working set assumption and according to the 80/20 rule - both are the key drivers of data placement - (80% of all accesses refers to 20% of the data – as in OLTP enterprise workloads [13]) statistics can be used during an update to inherit access frequencies to the new tuple version.

In SIAS, the length of the chain describes the amount of updates to a data item (amount of tuple versions). Hence, a long chain is correlated to a frequently updated data item. A page containing frequently updated tuple versions will likely contain mostly invisible tuples after some runtime, hence simplifying the merge/reclamation process.

Version Meta Data Placement: Version metadata embodying a tuple's visibility/validity is stored on the tuple itself in existing MVCC implementations. An update creates a new version and version information of the predecessor has to be updated accordingly. SIAS benefits largely from the avoidance of the in-place invalidation. Further decoupling visibility information and raw data would be even more benefitial. Raw data becomes stale and redundancies caused by, e.g., tuples that share the same content but different visibility information are reduced or vanish completely. A structure that separately maintains all visibility information, enables accessing only needed data (payload) on Flash memory. This principle inherently deduplicates tuple data and creates a dictionary of tuple values. Visibility meta-information can be stored in a column-store oriented method, where visibility information and raw tuple data form a n:1 relation. This facilitates usage of compression and compactation techniques. A page containing solely visibility meta-information can be used to pre-filter visible tuple versions which subsequently can be fetched in parallel utilizing the inherent SSD parallelism, asynchronous I/O and prefetching.

C. Optimizations

A number of optimization techniques can be derived from observation that in append based storage a page is never updated, yet: compression, optimization for cache and scan efficiency, page layout transformation etc. Generally these facilitate analytical operations (large scans and selections) on OLTP systems supporting archival of older versions.

Compression. Most DBMS store tuples of a relation exclusively on pages allocated for that very relation. In a multi version environment, versions of tuples of that relation are stored on a page. Since all these have the same schema (record format) and differ on few attribute values at most, the traditional light-weight compression techniques (e.g., dictionary- and run-length encoding) can be applied.

Page-Layout and Read Optimizations. Since the content of a written page is immutable and only read operations can access the page, a number of optimizations can be considered. If large scans (e.g. log analysis) are frequent, cache efficiency becomes an issue hence the respective page-layouts can be selected. Furthermore it is possible to use analytical-style page layout (e.g., PAX) for the version data and traditional slotted pages for the temporary or update intensive data such as indices.

V. Conclusion and Future Work

We propose the combination of multi-version databases and append-based storage as most beneficial to exploit new storage media. We have prototypically implemented SIAS in PostgreSQL and validated the reported simulation results. The highest performance benefit can be achieved by the integration of the append storage principle directly into a multi-version DBMS, reducing the update granularity to a tuple-version, implementing all writes out-of-place as appends, and coupling space management to version visibility. In contrast page remapping append storage manager does not fully benefit of the new storage technology. SIAS is a Flash-friendly approach to multi-version DBMS: (i) it sequentialises the typical DBMS write patterns, and (ii) reduces the net amount of pages written. The former has direct performance implications the latter has long-term longevity implications. In addition SIAS introduces new aspects to data placement making it an important research area. We especially identify version archiving, selection of hot/cold tuple versions, separation of version data and version meta-data, compression and indexing as relevant research areas.

In our next steps we focus on optimizations such as compression of tuple versions to further reduce write overhead by 'compacting' appended pages, placement of correlated tuple versions to increase cache efficiency as a 'per page clustering' approach and an efficient indexing of multi-version data using visibility meta-data separation.

Acknowledgment

This work was supported by the DFG (Deutsche Forschungsgemeinschaft) project "Flashy-DB".

References

[1] R. Gottstein, I. Petrov and A. Buchmann, "SIAS: Chaining Snapshot Isolation and Append Storage," submitted.

[2] A. Chan, S. Fox, W.-T. K. Lin, A. Nori, and D. R. Ries, "The implementation of an integrated concurrency control and recovery scheme," in *19 ACM SIGMOD Conf. on the Management of Data, Orlando FL*, Jun. 1982.

[3] I. Petrov, R. Gottstein, T. Ivanov, D. Bausch, and A. P. Buchmann, "Page size selection for OLTP databases on SSD storage," *JIDM*, vol. 2, no. 1, pp. 11–18, 2011.

[4] P. Bober and M. Carey, "On mixing queries and transactions via multiversion locking," in *Proc. IEEE CS Intl. Conf. No. 8 on Data Engineering, Tempe, AZ*, feb 1992.

[5] R. Stoica, M. Athanassoulis, R. Johnson, and A. Ailamaki, "Evaluating and repairing write performance on flash devices," in *Proc. DaMoN 2009*, P. A. Boncz and K. A. Ross, Eds., 2009, pp. 9–14.

 ISBN: 978-1-61208-247-9

[6] P. A. Bernstein, C. W. Reid, and S. Das, "Hyder - A transactional record manager for shared flash," in *CIDR*, 2011, pp. 9–20.

[7] M. Rosenblum, "The design and implementation of a log-structured file system," U.C., Berkeley, Report UCB/CSD 92/696, Ph.D thesis, Jun. 1992.

[8] M. J. Carey and W. A. Muhanna, "The performance of multiversion concurrency control algorithms," *ACM Trans. on Computer Sys.*, vol. 4, no. 4, p. 338, Nov. 1986.

[9] D. Majumdar, "A quick survey of multiversion concurrency algorithms."

[10] R. Gottstein, I. Petrov, and A. Buchmann, "SI-CV: Snapshot isolation with co-located versions," in *in Proc. TPC-TC*, ser. LNCS. Springer Verlag, 2012, vol. 7144, pp. 123–136.

[11] J. Krueger, C. Kim, M. Grund, N. Satish, D. Schwalb, J. C. H. Plattner, P. Dubey, and A. Zeier, "Fast updates on read-optimized databases using multi-core CPUs," in *Proceedings of the VLDB Endowment*, vol. 5, no. 1, sep 2011.

[12] M. Stonebraker, L. A. Rowe, and M. Hirohama, "The implementation of postgres," *IEEE Trans. on Knowledge and Data Eng.*, vol. 2, no. 1, p. 125, Mar. 1990.

[13] S. T. Leutenegger and D. Dias, "A modeling study of the TPC-C benchmark," in *Proc. ACM SIGMOD Conf.*, Washington, DC, May 1993, p. 22.

 ISBN: 978-1-61208-247-9