SIAS-V in Action: Snapshot Isolation Append Storage – Vectors on Flash (Extended Abstract)

Robert Gottstein #1, Thorsten Peter #1, Ilia Petrov #2, Alejandro Buchmann #1

#1 *Databases and Distributed Systems Group TU-Darmstadt,* #2 *Data Management Lab, Reutlingen University*

#1{gottstein | tpeter | buchmann}@dvs.tu-darmstadt.de, #2ilia.petrov@reutlingen-university.de

ABSTRACT

Multi-Version Database Management Systems (MV-DBMS) are wide-spread and can effectively address the characteristics of new storage technologies such as Flash, yet they are mainly optimized for traditional storage. A modification of a tuple in a MV-DBMS results in a new version of that item and the invalidation of the old version. Under Snapshot Isolation (SI) the invalidation is performed as an in-place update, which is suboptimal for Flash. We introduce *Snapshot Isolation Append Storage – Vectors (SIAS-V)*, which avoids the invalidation related updates by organising tuple versions as a simple linked list and by utilizing bitmap vectors representing different states of a single version. SIAS-V sequentializes writes and reduces the write-overhead by appending in tuple-version granularity, writing out only completely filled pages, and eliminating in-place invalidation.

In this demonstration we showcase the SIAS-V implementation in PostgreSQL side-to-side with SI. Firstly, we demonstrate that the I/O distribution of PostgreSQL under a TPC-C style workload, exhibits a dominant small-sequential write pattern for SIAS-V, as opposed to a random write dominated pattern under SI. Secondly, we demonstrate how the dense packing of tuple-versions on pages under SIAS-V reduces significantly the amount of data written. Thirdly, we show that SIAS-V yields to stable write performance and low transaction response times under mixed loads. Last but not least, we demonstrate that SIAS-V also provides performance improvements for traditional HDDs.

1. INTRODUCTION

Multi-Version approaches to database systems correlate well to the asymmentric properties of new storage technologies such as Flash. On asymmetric storage reads (especially random ones) are much faster than writes, compared to symmetric storage where both perform equally well. Under Multi-Version Concurrency Control (MVCC) or its subclass Snapshot Isolation (SI) [1] reads are never blocked; hence advantage of the high read performance can be taken. Writes produce a new physical version of the data item, which can be written out of place, yielding more Flash-friendly I/O patterns.

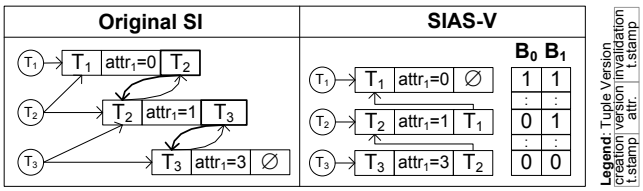

Figure 1: Version Organization in SI and SIAS-V

MV-DBMS introduce more complexity incurred by the version management overhead: (i) on the read side it relates to validity and visibility checks; (ii) on the write side it relates to version organisation and write I/O. Read operations fetch the latest version of the data item, a transaction is allowed to see. The version visibility is determined using timestamps based on transactional time. Each version maintains a creation and an invalidation timestamp, which correspond to the ID of the creating/modifying transaction. Upon an update of a data item a new version is created/inserted and the old version is invalidated by stamping it with an invalidation timestamp. Under traditional SI both the old (invalidated) and the newly created version need to be written back (the former 'in-place'), causing Flash-adverse random write patterns.

In this demonstration we showcase a new approach called SIAS-V, which algorithmically avoids in-place invalidation of versions by creating a simple linked list of all versions of a data item and additionally storing a visibility and validity bit for each version using bitmap vectors. SIAS-V addresses the read/write asymmetry of Flash, leveraging their fast random read rate and circumventing overwrites. Figure 1 depicts a short example (extended in Section 3): three transactions T_1, T_2 and T_3 update tuple a, creating versions a_0, a_1 and a_2 respectively. While under original SI, T_3 modifies a_2 and a_1, under SIAS-V T_3 creates and appends a_2, and also sets the bit values in the newly introduced bit vectors B_0 and B_1 indicating visibility and validity respectively.

2. RELATED WORK

The general Snapshot Isolation (SI) algorithm is introduced and discussed in [1]. Specifics of a concrete SI implementation (PostgreSQL) are described in detail in [6, 11]. The general SI algorithm [1] assumes a logical version or-

ganisation as a double-linked list and the maintenance of creation and invalidation timestamps on the data-item version itself, while making no assumption about the physical organisation. An improvement of SI called SI-CV, colocating versions per transactions on pages has been proposed in [5]. SIAS-V organizes data item versions in simple chronologically ordered backwards-linked chains. Alternative approaches have been proposed in [3, 8] and explored in [9, 2] in combination with MVCC algorithms and special locking approaches. The high-performance MVCC algorithm employed in Hekaton [8] assumes that a creation and invalidation timestamp are maintained on every version and utilises both to perform visibility checks or correct validation upon transaction commit. We assume that SIAS-V is applicable in such settings.

Similar chronological-chain version organization has been proposed in the context of update intensive analytics [7]. In such systems data-item versions are never deleted, instead they are propagated to other levels of the memory hierarchy such as hard disks or Flash SSDs and archived. Any logical modification operation is physically realized as an append. SIAS-V on the other hand provides mechanisms to couple version visibility to (logical and physical) space management. Another difference is that SIAS-V uses transactional time (all timestamps are based on a transactional counter) in distinction to timestamps that correlate to logical time (dimension). Stonebraker et al. realized the concept of TimeTravel in PostgreSQL [12]. In [4] we evaluate SIAS and append based storage management approaches on Flash.

3. SIAS-V ALGORITHM

This section provides a short overview of the SIAS-V algorithm and prototype, implemented by modifying an out of the box PostgreSQL 9.0.4.

Individual tuple versions are appended to a page of the corresponding relation, which is either written after it gets full or after a threshold is reached. SIAS-V uses the same steal–no-force approach as PostgreSQL and no additional retaining mechanisms are employed to keep newly added pages in the buffer. In the following we discuss the simplified visibiliy check in SIAS-V as depicted in Algorithm 1. It determines the visibility of a tuple version X_v for a transaction T. In Snapshot Isolation the visibility check is seen as a local decision (per tuple version). SIAS-V introduces a change of paradigm and decides on the data item as a whole (all tuple versions).

The validity bitmap vector which is denoted by B_1 is mandatory for the SIAS-V algorithm. It logically partitions the database into valid and invalid tuple versions. A tuple version is considered invalid if and only if it has a valid (committed) successor. The most recent valid version of each tuple chain is called the *entrypoint* and serves as a starting point for the visibility check. In order to find a visible version, the corresponding chain is traversed backwards. As soon as the visible version is found, the chain traversal can be interrupted. Bitmap vector B_0 is optional and represents an optimization that maintains visibility information (dead tuples) and acts as a tuple version mask, filtering tuple versions which are invisible to all running transactions.

For each version a corresponding entry in B_0 and B_1 exists. The check immediately returns true if the transaction itself created the tuple version (line 1). If the version is marked as invisible in B_0, the check returns $false$ (line 3). If the transaction's ID is smaller than the tuple version's creation ID then the version is not visible (line 4). If the version's ID is smaller than the transaction's ID (line 6) the tuple version has to be valid and committed to be visible (line 7). If the tuple version is not valid it may be visible if the committed successor in the chain is not visible (successor was created after the start of the transaction) else it is invisible (lines 10,12,13).

Algorithm 1 SIAS-V Visibility Check

```
1: if (X.txmin == t.tid) then
2:     return true;
3: else if (!B0[X.id]) then
4:     if (X.txmin > T.id) then
5:         return false;
6:     else if (X.txmin < T.id) then
7:         if (!B1[X.id] AND isCommitted(X.txmin)) then
8:             return true;
9:         else
10:            Tuple XNew = getSuccessorVersion(X);
11:            if (XNew.txmin < T.tid AND
12:            isCommitted(XNew))
13:            return false; else return true; end if
14:        end if
15:    end if
16: end if
17: return false;
```

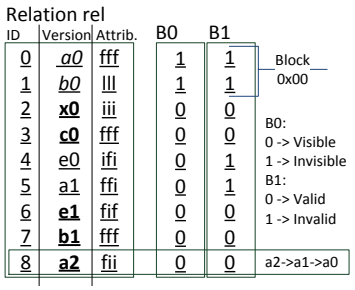

Figure 2: Bitmap Vectors with Tuple Versions

A fetch of *a successor version* is the most costly operation and involves searching - this case can only occur when accessing an invalidated tuple directly (not starting with the *entrypoint*). SIAS-V adapts access methods to avoid such accesses as discussed in Section 3.1.

Example: Figure 2 shows table *rel* with both bitvectors. The table keeps 5 data items, represented by 9 versions, each containing one attribute. E.g. data item a exists in three versions: $\{a0; a1; a2\}$. Version $a0$ is not visible to any running transaction, $a1$ is visible to transactions which cannot see the most recent version $a2$. Each data item has one *entry point*: $\{a2; b1; e1; c0; x0\}$; indicated by a zero bit in B_1.

Tuple versions of a relation are appended to a page of that relation. It can be written to stable storage as soon as it is filled or a pre-defined threshold is reached. A page is immutable once it is written (thus it can be garbage collected once it contains only invalid and invisible versions).

Size: Each entry in each vector needs one bit and the position of the bit equals the tuple versions' ID. Hence one

8KB page is able to maintain 64K entries.

3.1 Operations: Read, Update, Delete

Read: Before a table scan is started a virtual snapshot of B_1 is created. A subsequent merge is unnecessary since the scan is read only. B_1's copy therefore snapshots the relation at the time the scan starts. B_0 is not copied since changes to it depend on the oldest still running transaction, hence false negatives (not visible) are avoided, but false positives may exist, which are filtered during the visibility check using B_1. B_0 is beneficial since versions with their B_0 bit set can be discarded (need not be read or processed). Valid tuple versions, indicated by a zero in B_1, are the *entrypoint* for a traversal of the versions representing a data item in different states. Each version maintains a backlink to and the creation timestamp of its predecessor (see Figure 1). From each valid version the simplified visibility check (Algorithm 1) is executed. Since the *entrypoint* is always valid no visible successor version exists at that time and line 10 can never be reached, which is also true when traversing the chain backwards. If the check returns false and there exists a predecessor version p_0 to some tuple P it may be visible. p_0 only has to be fetched and checked if P does not store the creation timestamp of p_0. E.g. during a scan of *rel* (Figure 2) SIAS-V accesses the entry points using B_1. Block $0x00$ is not read since all contained versions are not visible (indicated by B_0). When *entrypoint* a_2 is read, the visibility check is executed. In the worst case a_2's predecessor a_1 has to be read. In contrast, original snapshot isolation fetches block 0x00 and checks each tuple individually.

Update/Insert: An update is executed as an invalidation and an insertion of a new version. To invalidate a version, the respective bit in B_1 is set. At the same time the new version and its corresponding values in both vectors are inserted. The initial values of the new version are set to zero in both vectors. The new version receives its own creation timestamp and maintains a pointer to its predecessor and the predecessor creation timestamp. The tuple is stored in a page which is written after it is completely filled or a threshold is reached.

Delete: A deletion is treated as an invalidation and an insertion of a special tombstone version indicating the deletion and terminating the version chain. The compensation version becomes the visible version and serves as the entrypoint. When it becomes the only visible version, the bits are set in B_1 and B_0 to render all versions of the data item invisible - such that no access to the tuple is needed in order to discard it during the visibility check.

4. DEMONSTRATION

After introducing the audience to the basics of SIAS-V they select a demonstration scenario out of a set described in more detail below.

4.1 Invalidation Model and I/O Distribution

Conceptually, SIAS-V introduces a new version organisation and invalidation model. It leads to SSD-friendly write patterns by significantly reducing random writes and utilising out of place updates (Figure 3).

To demonstrate this claim we use a tool called SIAS Analyzer (Figure 4) to compare SI and SIAS-V in PostgreSQL under TPC-C (DBT-2) style workload and to visualize disc accesses. The audience is shown which blocks of which relation are written at the current time and at which I/O rate the requests are processed. These are visualized and the spatial and the temporal distribution of writes is depicted (Figure 3) to underline our claims. We also provide a short video illustrating the SIAS Analyser [10]

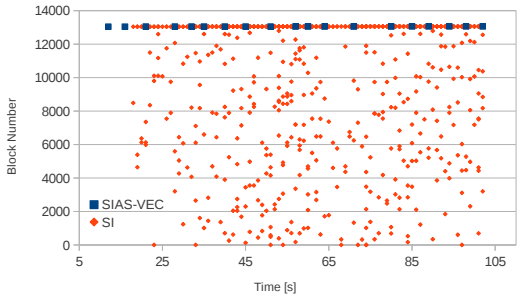

Figure 3: Spatial Distribution of Write I/Os under SI and SIAS-V. A 100 sec. slice of the TPC-C run.

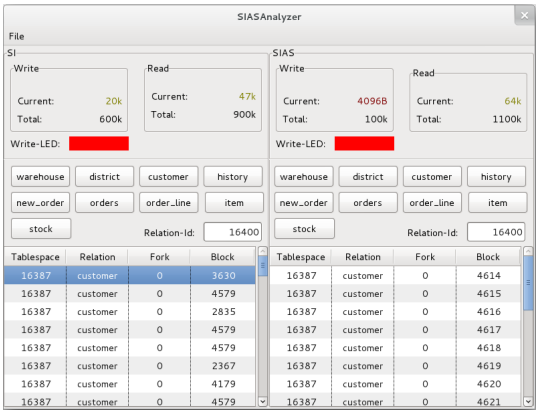

Figure 4: Screenshot SIAS Analyzer

4.2 Dense packing and endurance

Due to the SIAS-V invalidation model, new versions can be densely packed on pages in SIAS-V which leads to less SSD writes that can also be performed sequentially thus increasing Flash endurance.

To demonstrate this claim we use the SIAS Analyzer (Figure 4) and let the audience pick a time frame from the TPC-C run performed in Scenario 4.1. Next we show the writes performed by SIAS-V. The audience can observe the sequential nature of the I/O distribution as well as why this yields lower Flash block-erase count and increased endurance.

4.3 Write Performance and Low Response Times

SIAS-V offers stable write throughput. It isolates the I/O read and write streams on an SSD. A Flash SSD device can deliver predictable performance under mixed workloads.

We demonstrate this claim by defining a *One-Tuple-Update-Micro-Benchmark*. It updates a single tuple on every page the relation occupies. This corresponds to the worst case update predicate touching a sparse subset of the relation. In real world terms this corresponds to a budget increase of all departments that do not exceed a certain maximum budget. Under SI a significant overhead is incurred when a few tuples per page get invalidated since the whole page is marked dirty and is written back in-place causing expensive random writes.

Under SIAS-V new versions are densely packed (Scenario 4.2) and appended sequentially. Under SI previous tuple versions are invalidated yielding unpredictable mixed I/O load (old versions are random read and random written after invalidation). During the benchmark run the audience can vary the number of transactions and experiment with the benchmark runtime (Figure 5). Based on the high standard deviation (visualized by the error-bars in Figure 5) the audience will experience the performance instability of SI for the selected number of concurrent transactions.

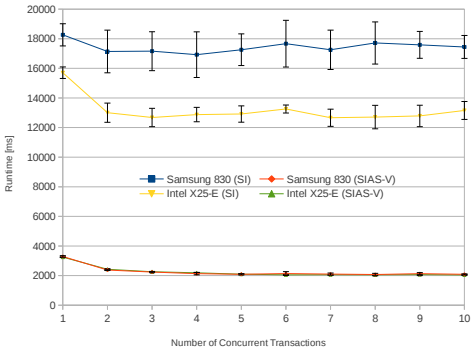

Figure 5: One-Tuple-Update-Micro-Benchmark: Avg. runtimes for different benchmark runs and the standard deviations as error-bars.

SIAS-V exhibits low response times per transaction for write-intensive and/or mixed-workloads. Response times under SI tend to exhibit higher workload-dependent variance.

We illustrate this claim again using the *One-Tuple-Update-Micro-Benchmark*. Since it updates one tuple per page occupied by the table, it practically implies the following. If a relation comprises 100 pages and a page contains 200 tuple versions SI will update 100 pages due to version invalidation and will add a single new page for the new versions. Conversely, SIAS-V, will just append the new versions possibly adding a new page.

The audience will be able to vary the number of concurrently executing transactions to experience (as shown in Figure 5) that: (i) the response times under SIAS-V are significantly lower than under SI; (ii) response times under SIAS-V are stable. Furthermore, the *One-Tuple-Update-Micro-Benchmark* shows the effect of write reduction and dense packing.

4.4 Dense Packing and Less Read I/O

Under read-only workloads SIAS-V achieves lower scan times than SI since it requires less read I/O due to SIAS-V dense packing and version organisation.

We demonstrate this claim with a *read-only-benchmark* that is executed on the DB state after Scenario 4.3. Several tables in PostgreSQL are concurrently updated by one transactions each (all tables have non-overlapping storage areas). The audience will observe the lower completion times of SIAS-V.

4.5 Advantages on traditional HDDs

SIAS-V also leads to better results on traditional HDDs. This claim is again demonstrated by running the *One-Tuple-Update-Micro-Benchmark* on an HDD. As Figure 6 shows SIAS-V has performance advantage over SI due to write reduction and avoidance of random writes. On SSDs however the performance benefits of SIAS-V are significantly higher (compare Figures 5 and 6).

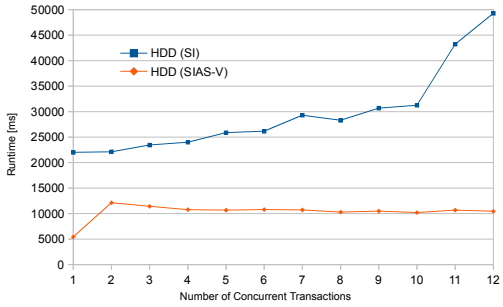

Figure 6: One-Tuple-Micro-Benchmark on HDD.

5. CONCLUSIONS

In this paper we demonstrate SIAS-V – an approach to multi-version organisation that algorithmically avoids in-place invalidation of versions by creating a simple linked list of all versions of a data item and by additionally storing a visibility and validity bit for each version using bitmap vectors. SIAS-V: (i) introduces a new invalidation model and version organisation; (ii) sequentialises write I/Os; (iii) packs new versions densely on pages and reduces the total amount of writes; (iv) yields stable write performance and low response times.

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

6. REFERENCES

[1] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ansi sql isolation levels. *SIGMOD Rec.*, 24(2):1–10, May 1995.

[2] P. Bober and M. Carey. On mixing queries and transactions via multiversion locking. In *ICDE*, 1992.

[3] A. Chan, S. Fox, W.-T. K. Lin, and et al. The implementation of an integrated concurrency control and recovery scheme. In *Proc. SIGMOD*, 1982.

[4] R. Gottstein, I. Petrov, and A. Buchmann. Append storage in multi-version databases on flash. In *Proc. of BNCOD 2013*.

[5] R. Gottstein, I. Petrov, and A. Buchmann. SI-CV: Snapshot isolation with co-located versions. In *Proc. TPC-TC*. 2012.

[6] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *VLDB*, 2000.

[7] J. Krueger, C. Kim, M. Grund, N. Satish, D. Schwalb, and et al. Fast updates on read-optimized databases using multi-core cpus. In *Proc. VLDB*, 2011.

[8] P.-A. Larson, S. Blanas, C. Diaconu, and e al. High-performance concurrency control mechanisms for main-memory databases. *Proc. VLDB*, 2011.

[9] I. Petrov, R. Gottstein, T. Ivanov, D. Bausch, and A. P. Buchmann. Page size selection for OLTP databases on SSD storage. *JIDM*, 2(1):11–18, 2011.

[10] SIAS Analyser demonstration video (5MB). http://dblab.reutlingen-

university.de/tl_files/downloads/SIAS_V-InAction.mp4.

[11] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 2011.

[12] M. Stonebraker, L. A. Rowe, and M. Hirohama. The implementation of postgres. *TKDE*, 2(1):125, 1990.