Append Storage in Multi-Version Databases on Flash

Robert Gottstein[1], Ilia Petrov[2], and Alejandro Buchmann[1]

[1] TU-Darmstadt, Databases and Distributed Systems, Germany
{gottstein,buchmann}@dvs.tu-darmstadt.de
[2] Reutlingen University, Data Management Lab, Germany
ilia.petrov@reutlingen-university.de

Abstract. Append/Log-based Storage and Multi-Version Database Management Systems (MV-DBMS) are gaining significant importance on new storage hardware technologies such as Flash and Non-Volatile Memories. Any modification of a data item in a MV-DBMS results in the creation of a new version. Traditional implementations, physically stamp old versions as invalidated, causing in-place updates resulting in random writes and ultimately in mixed loads, all of which are suboptimal for new storage technologies. Log-/Append-based Storage Managers (LbSM) insert new or modified data at the logical end of log-organised storage, converting in-place updates into small sequential appends. We claim that the combination of multi-versioning and append storage effectively addresses the characteristics of modern storage technologies.

We explore to what extent multi-versioning approaches such as Snapshot Isolation (SI) can benefit from Append-Based storage, and how a Flash-optimised approach called SIAS (Snapshot Isolation Append Storage) can improve performance. While traditional LbSM use coarse-grain page append granularity, SIAS performs appends in tuple-version granularity and manages versions as simply linked lists, thus avoiding in-place invalidations.

Our experimental results instrumenting a SSD with TPC-C generated OLTP load patterns show that: a) traditional LbSM approaches are up to 73% faster than their in-place update counterparts; b) SIAS tuple-version granularity append is up to 2.99x faster (IOPS and runtime) than in-place update storage managers; c) SIAS reduces the write overhead up to 52 times; d) in SIAS using exclusive append regions per relation is up to 5% faster than using one append region for all relations; e) SIAS I/O performance scales with growing parallelism, whereas traditional approaches reach early saturation.

1 Introduction

Multi-Version Database Management Systems (MV-DBMS) and Log/Append-based Storage Managers (LbSM) are gaining significant importance on new storage hardware technologies such as Flash and Non-Volatile Memories. Compared to traditional storage such as HDD or main memory new storage technologies have fundamentally different characteristics. I/O patterns have major influence on their performance and endurance: especially overwrites and (small) random writes are significantly more expensive than a sequential write.

MV-DBMS create new versions of data items once they are modified. Treating old and new versions differently provides a mechanism to leverage some of the properties of

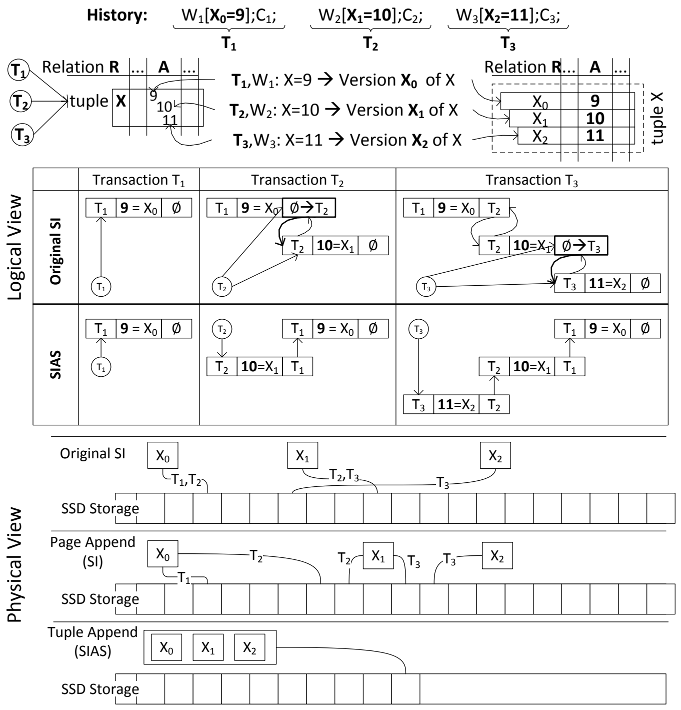


Fig. 1. Version handling

new storage, such as fast reads and low latency. However, HDD (read) optimised implementations such as Snapshot Isolation (SI) invalidate old versions physically in-place as successor versions are created, resulting in random writes and mixed load which is suboptimal for new storage technologies. Additionally they do not leverage read/write asymmetry. Log/Append-based storage managers (LbSM) organise the storage as a circular log. They physically append modified/new data at the end (the logical head) of the log, which eliminates in-place updates and random writes. LbSM maintain a mapping of appended blocks and pages, they do not address issues related to version organisation such as additional write overhead introduced by the in-place invalidation.

We claim that the combination of a MV-DBMS using LbSM effectively addresses the characteristics of modern storage technologies. We further state that the most promising approach for append storage needs to be implemented *within the architecture and algorithms* of modern MV-DBMS. The following example offers a detailed description. Fig. 1 shows the invalidation process of different MV-DBMS (SI, SIAS), coupled to different types of storage managers ('in-place update' as original SI, page granularity LbMS, tuple granularity LbSM): three Transactions ($T1$, $T2$, $T3$) update data item X in serial order resulting in a relation that contains three different tuple versions of data item X. $T1$ creates the initial version X_0 of X. $T2$ issues the first update. In original SI, X_0 is invalidated in-place by setting its invalidation timestamp, subsequently X_1 is created. The update issued by $T3$ proceeds analogously and X_1 is invalidated in-place while X_2 is created as a new tuple version. Original SI, coupled to in-place update storage manager, writes X_0 and X_1 to the same location (random write) after the updates $T1$ and $T2$ (the initial page based on the free space). Original SI, coupled to a page append LbSM, will write X_0 and X_1 to pages with a higher page number (small sequential append). The payload of the write (updated versions/total versions per page) may be very low, yielding 'sparse' writes. Under SIAS X_0, X_1 and X_3 will be buffered, placed on the same page and appended altogether.

The *contributions* of this paper are as follows. We explore the performance implications of an approach combining LbSM and MV-DBMS optimised for Flash storage called SIAS (Snapshot Isolation Append Storage).

It organises versions in MV-DBMS as a simple backwards-linked list and assigns all versions of a data item a virtual ID. SIAS involves adapted algorithmic version invalidation handling and visibility rules. Furthermore, it is natively coupled to a LbSM and uses tuple granularity for logical append I/Os.

The *experimental results* under a TPC-C aligned OLTP load show that: a) 'page append LbSM' is up to 73% faster than traditional the 'in-place update' approach; b) using SIAS version-wise append we observe up to 2.99 times improvement in both IOPS and runtime; c) SIAS reduces the write overhead up to 52x; d) page-append LbSM yields equal write amount as the 'in-place update' approach; e) space reclamation due to deleted/invisible tuples is not suitable for append LbSMs in general and slows them down by approx. 40%; f) in SIAS using one local append region per relation is up to 5% faster than one global append region; g) using page remapping append with one global region is approx. 4.5% faster than using a local region; h) all append storage I/O performance scales with growing parallelism where in-place update approaches reach early saturation.

The paper is organised as follows: Section 2 provides a brief overview on related algorithmic approaches and systems; a general introduction of the used algorithms (SI and SIAS) is provided in Section 4; the main characteristics of Flash storage are summarised in Section 3. Section 2 describes combinations of in-place and append storage managers. Our experimental setup and framework are described in Section 6. The experimental results are discussed in Section 7.

2 Related Work

Snapshot Isolation (SI) is introduced and discussed in [2]. Specifics of a concrete SI implementation (PostgreSQL) are described in detail in [24,20]. As reported in [2] SI fails to enforce serializability. Recently a serializable version of SI was proposed [5] that is based on read/write dependency testing in serialization graphs. Serializable SI assumes that the storage provides enough random read throughput needed to determine the visible version of a tuple valid for a timestamp, making it ideal for Flash storage. [19] represents an alternative proposal for SI serializability. In addition serializable SI has been implemented in the new (but still unstable) version of PostgreSQL and will appear as a standard feature in the upcoming release.

SI [2] assumes a logical version organisation as a double-linked list and a two place invalidation, while making no assumption about the physical organisation. An improvement of SI called SI-CV, co-locating versions per transactions on pages has been proposed in [10].

Alternative approaches have been proposed in [7] and explored in [17,4] in combination with MVCC algorithms and special locking approaches. [17,4,7,11] explore a log/append-based storage manager. A performance comparison between different MVCC algorithms is presented in [6]. [15] offers insights to the implementation details of SI in Oracle and PostgreSQL. An alternative approach utilising transaction-based tuple collocation has been proposed in [10].

Similar chronological-chain version organisation has been proposed in the context of update intensive analytics [14]. 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 realised as an append. SIAS on the other hand provides mechanisms to couple version visibility to (logical and physical) space management. Another difference is that SIAS uses transactional time (all timestamps are based on a transactional counter) as opposed to timestamps that correlate to logical time (dimension). Stonebraker et al. realised the concept of TimeTravel in PostgreSQL [22].

Multi-Version Database Systems. While Time-travel and MVCC approaches have been around for three decades, MV-DBMS approaches are nowadays applied in in-memory computing systems such as Hyper [13] or HYRISE [12] to handle mixed OLAP, OLTP loads, to handle database replication (Postgre-R) etc.

MV-DBMS are a good match for enterprise loads [14]. As discussed in [14], these are read-mostly; the percentage of writes is as low as approx. 17% (OLTP) and approx. 7% (OLAP) [14]. Since reads are never blocked under MVCC, in such settings there are clear performance benefits for the read-mostly enterprise workloads.

Multi-version approaches are widely spread in commercial and open source systems. Some MV-DBMS systems are: Berkeley DB (Oracle), IBM DB2, Ingres, Microsoft SQL Server 2005, Oracle, PostgreSQL, MySQL/InnoDB. And in addition *in-memory* systems such as Hyper [13], Hyder [3] etc.

Multi-Version approaches and MV-DBMS leverage the properties of new hardware. In this paper we investigate how these can be utilised to leverage I/O asymmetry of new storage technologies. Multi-version approaches can be used to leverage hardware characteristics of modern CPUs in transparently creating snapshots of in-memory pages [13] or to control data placement and caching in memory hierarchies.

Append Storage Management. LbSMs follow the principle of appending new data at the end of log structured storage. MV-DBMS alleviate appending of new data in principle, yet traditional approaches write data to arbitrary positions, updating data in-place or allocating new blocks. Such *traditional approaches*, implemented in current databases, address special properties of HDDs – especially their high sequential throughput and high latency access time on any type of I/O. They maintain clustering by performing in-place updates to optimise for read accesses, reducing the latency introduced by the properties of HDDs (rotational delay, positioning time). Thus implementations like SI in PostgreSQL rely on the in-place invalidation of old tuple versions. New storage technologies introduce fundamentally different properties (Section 3) and require optimised access methods. Especially low latency access time and fast random reads are not addressed yet and have to be leveraged.

LbSMs address the high throughput of large sequential writes on HDDs but destroy clustering, since new and updated data is not clustered with existing data yielding the same clustering attributes. Approaches using delta stores still require relatively expensive merge operations and generate overhead on read accesses [16].

The applicability of LbSMs for novel asymmetric storage has been partially addressed in [21,3] using page-granularity, whereas SIAS employs tuple-granularity (tuple append LbSM) much like the approach proposed in [4], which however invalidates tuples in-place. Given a page-append LbSM the 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.

3 Flash Memories

Enterprise Flash SSDs independent of their hardware interfaces (SATA, PCIe), exhibit significantly better performance and very different characteristics than traditional hard disks. Since most DBMS were build to compensate for the properties of HDDs, they tread SSDs as HDD replacement, which yields suboptimal performance. The most important characteristics of Flash are:

(i) Asymmetric read/write performance – reads are up to an order of magnitude faster than writes as a result of the physical NAND properties and their internal organisation. NAND memories introduce erase as an additional third operation together with read and write. Before performing a write, the whole block containing the page to be written must be erased. Writes should be evenly spread across the whole volume to avoid damage due to wear and increase endurance - wear-levelling. Hence no write in-place as on HDDs, instead copy-and-write. (ii) High random read performance (IOPS) – random reads for small block sites are up to hundred times faster than on an HDD. (iii) Low random write performance (IOPS) – small random writes are five to ten times slower than reads. Random writes depend one the fill-degree of device and incur a long term performance degradation due to Flash-internal fragmentation effects. (iv) Good sequential read/write transfer. Sequential operations are asymmetric, due to techniques as read ahead and write back caching the asymmetry is below 25%. (v) Suboptimal mixed load performance – Flash SSDs can handle pure loads (read or write) very well despite of the degree of randomness (random writes excluded). (vi) Parallelism – Compared to the typical hard drive and due to their multi-chip internal organisation Flash can handle much higher levels of I/O parallelism, [8],[1].

4 Algorithmic Description

In the following section we give a brief introduction to the SI algorithm as originally proposed in [2]. We then illustrate SI by using the implementation in PostgreSQL and point out differences and optimisations. Finally we present the SIAS algorithm.

4.1 Snapshot Isolation Introduction

SI is a timestamp based MVCC mechanism which assigns each running transaction exactly one timestamp and each data item two. The transaction's timestamp corresponds to the start of the transaction and the data item's timestamps correspond to the creation, respectively the invalidation of that item. An invalidation is issued on an update/deletion of an item. Each running transaction executes against its own version/snapshot of the committed state of the database. Isolated from effects of other concurrently running transactions, a transaction is allowed to read an older committed version of a data item instead of reading a newer, uncommitted version of the same item or being blocked/aborted. A snapshot describes the visible range of items the transaction is able to "see" (facilitated by the timestamps). On an access to an item the transaction's timestamp is compared to the ones on the item. Items with a higher creation timestamp (inserted after the start of the transaction) are invisible and such with a lower (or equal) timestamp are visible to the transaction as long as they are committed and were not inserted concurrently. Reads are therefore never blocked by writes and changes made by a transaction are executed on its own snapshot which becomes visible to follow up transactions after its successful commit. Whether or not a commit is successful is determined at commit time by the transaction manager, which performs a write set comparison of the involved concurrent transactions. Overlapping write sets between concurrent transactions are not allowed and lead to the abort of at least one transaction since it is not allowed to have more than one update to an item. Two equivalent rules guarantee this behaviour: "first-committer-wins" [2] and "first-updater-wins" [2],[20]. The former corresponds to a deferred check at commit time, while the latter is enforced by immediate checks e.g. exclusive locks.

4.2 SIAS - Algorithm

Fig. 1 shows how different versions are handled under different approaches. SIAS [18] introduces a new addressing mechanism: (multiple) tuple versions are addressed as a chain by means of a virtual tuple ID (VID) that identifies the chain (as one logical item; all tuple versions in the chain share the same VID).

When a tuple-version is read the entry point is fetched first and the visibility can be determined for each VID. If the entry points timestamp is too high or equals a concurrent transaction, the predecessor version is fetched. Visibility information is coded within the algorithmic chain traversal access methods. Each version $n (n \neq 0)$ of a tuple is linked to its predecessor $n - 1$. The first version ($n = 0$) points to itself or uses a $NULL$ pointer. The VID identifies a chain; each member-tuple receives a unique tuple-ID (TID) as well as a *version count* that indicates its position within the chain. The newest member has the highest chain count and becomes the *entry point*. To speed up VID lookups an in-memory data structure, recording of all entry points is created (Sect. 4.3). The tuple structure used by SIAS is shown in Table 1 and illustrated in the following example. Assume two data items X and Y forming two chains; X was updated once and Y twice. The entry points are versions X_1 and Y_2 (marked bold in Table 1). Each version maintains a *pointer* to its predecessor forming a physical chain. The visibility is determined by a chain traversal, starting at the entry point applying

Table 1. SIAS - On-Tuple Information

Tuple	Creation Xmin	Predecessor Pointer	Predecessor Xmin (Xpred)	VID	Version Count
X0	15	X0	null	0x0	0
X1	38	X0	15	0x0	1
Y0	50	Y0	null	0x23	0
Y1	83	Y0	50	0x23	1
Y2	110	Y1	83	0x23	2

the SIAS algorithm rules – instead of reading an invalidation timestamp the creation timestamps of two subsequent versions are compared (xmin, xmin_pred).

SIAS verifies visibility of tuple versions based on the entry point, while SI inspects each tuple version individually. The number of chains equals the number of entry points (items) while the amount of tuple versions in the database can be much higher. SIAS entry-points represent a subset of all tuple-versions and at most one tuple-version per chain is visible to each transaction. The visibility control can discard a whole (sub-) chain of tuple-versions, depending on the value of the creation timestamp, thus saving I/O. Hence on average, SIAS has to read less tuple-versions to determine visibility, but may perform more read I/Os to fetch the appropriate version. The most recent version may not be the one visible for an older (longer running) transaction.

4.3 SIAS - Data Structures

SIAS introduces two data structures to organise the entry point information:

(i) $dstructI$: mapping of the VID to the chain member count.

(ii) $dstructII$: mapping of the VID to (the location of) the entry point (TID).

$dstructI$ accelerates verification of the following condition: is the tuple-version under inspection an entry-point or has the entry-point been changed by another committed transaction. This information can also be obtained by comparing the tuple ID (TID) of the read tuple version and the TID within $dstructII$, thus making $dstructI$ optional.

$dstructII$ is used to access the current entry-point.

The chaining mechanism based on VIDs has the following implications: (a) *The chain length* depends on whether old and invisible versions are retained/archived and on the update frequency and duration of transactions. The chain length is therefore defined by the total amount of updates on the respective tuple. (b) *The amount of extra reads* due to chain traversal depends on (c) *The amount of visible versions.*

5 Append Storage

In the following we briefly introduce our approaches to append storage in MV-DBMS. We classify the approaches in page-wise and tuple-wise LbSMs, further categorize them according to Figure 2 and explain them in more detail in the following sections.

5.1 Page-Append

The page-append LbSM describes append storage outside the database, without knowledge of the inertia of transmitted pages, implementing a 'holistic' page remapping paradigm. We utilise a default out of the box PostgreSQL running under a SI MVCC mechanism (Sect. 6), enhanced by LbSMs in the following variants:

SI-PG (SI – Page Global) denotes the traditional approach where pages are written to one single append region on the storage device; hence a global append region. It performs a remapping of page- and block addresses. We simulate variants with (SI-PG-V) and without a garbage collection mechanism (SI-PG-NV); we refer to SI-PG when describing both variants.

SI-PL (SI – Page Local) extends the SI-PG approach with multiple append regions. SI-PL receives more information about the content of the transmitted pages. SI-PL partitions the global append storage into multiple *local* append regions, dedicating each region to a single relation of the database. We simulate variants with (SI-PL-V) and without a garbage collection mechanism (SI-PL-NV); we refer to SI-PL when describing both variants.

PostgreSQL uses a space reclamation process called vacuum to garbage collect invisible versions (Sect. 5.4). SI-PG and SI-PL do not require changes to the MV-DBMS. They rather maintain a mapping of pages, performing block-address translation to generate *flash-aware* patterns. Both can be implemented as a layer between the device and the MV-DBMS. Although this already delivers some benefits for new storage media such as flash, our evaluation shows that those can be optimised by inherently integrating the storage manager into the MV-DBMS.

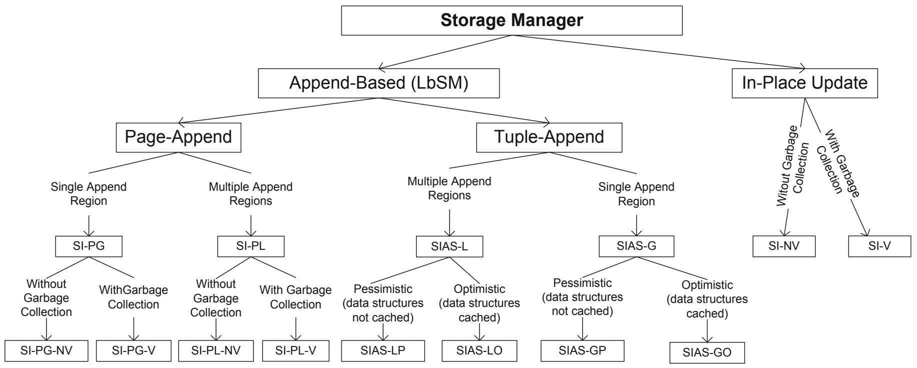


Fig. 2. Storage Manager Alternatives

5.2 SIAS - Tuple Append LbSM

We developed *SIAS (Snapshot Isolation Append Storage)* which algorithmically changes SI and improves on it by enabling tuple based appends without the need for in-place invalidation of old tuple versions. SIAS appends tuples to a page until it is filled and subsequently appends it to the head of the corresponding append log.

SIAS-L uses multiple append regions, where each region is dedicated to exactly one relation of the database. All pages in a local append region belong to the same relation and all tuples within a page belong to the same relation. Tuples are appended to a page of their relation, which is subsequently appended to the relation's local append region, after the page is filled completely or has reached a certain threshold.

SIAS-G uses one append region for all pages. Tuples within a single page belong to the same relation. Tuples get appended to a single page (analogously to SIAS-L) which is then appended to a global append region. The global append region maintains pages of all the relations of the MV-DBMS.

According to the SIAS concept we compare two variants of SIAS-L and SIAS-G, an *optimistic* approach which assumes that SIAS data structures are cached (SIAS-LO and SIAS-GO) and a *pessimistic* approach which fetches the data structures separately (SIAS-LP and SIAS-GP), thus resulting in four variants of SIAS. Since the test results of all SIAS variants showed the same performance independent of the garbage collection process, we omit the detailed report of these additional results in this paper and refer to SIAS-L and SIAS-G subsuming all four approaches.

5.3 In-Place - No Append

For the in-place approach we use the original Snapshot Isolation in two configurations:

SI-NV (SI No Vacuum) – deactivated garbage collection in PostgreSQL (vacuum),
SI-V (SI with Vacuum) – activated garbage collection (vacuum) in PostgreSQL.

5.4 Space Reclamation

In LbSMs written data is immutable, whereas in a MV-DBMS invalidated versions of tuples become invisible and occupy space which can be freed. The page-append LbSM has no knowledge about invalidated versions and therefore has to rely on methods within the MV-DBMS for space reclamation (e.g. vacuum in PostgreSQL).

Physical blocks get invalidated because the logical pages were remapped to another position and have to be physically deleted on the Flash device. The moment of execution is implementation dependent. On Flash an erase can only be performed in granularities of an erase unit - usually much larger than a page. Issuing an overwrite of a block (instead of deleting it) results in a remapping within the Flash device and therefore to unpredictable performance analogously to an in-place update (black box). Physical deletes should therefore only be issued in erase unit granularity (using trim). Pages which are still valid and reside within the unit which is about to be erased have to be re-mapped/re-inserted (append).

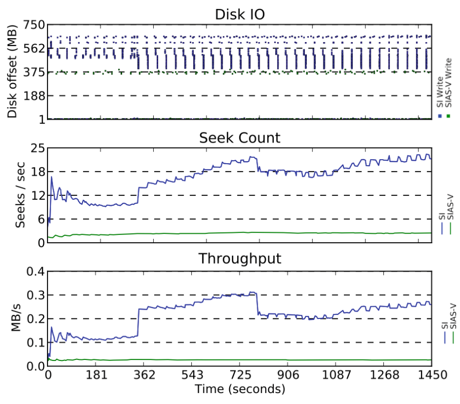
The tuple-append LbSM in SIAS is able to garbage collect single versions of a tuple. A page may contain still valid tuples which are simply re-inserted into the append log. Since each page is appended as soon as it is filled, the pages stay compact.

6 Evaluation

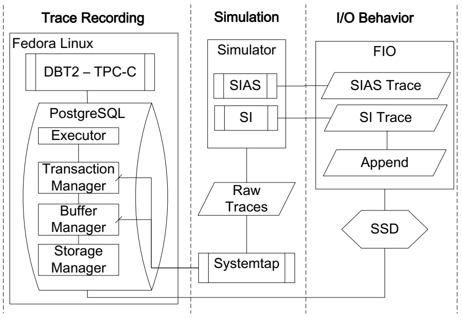
Our evaluation of the different LbSM alternatives (Sect. 2) is based upon a trace driven simulation, which we describe in the following paragraphs. We opted for simulation for two reasons: (a) to focus on the main characteristics of the multi versioning algorithms (i.e. exclude influences of the transaction-, storage- and buffer-manager as well as PostgreSQL's specific overhead); and (b) to compare a number of LbSM and SIAS alternatives. The simulator was validated against the PostgreSQL implementation of our baseline SIAS algorithm (see *validation* in this section). The simulation workload is created by an open source TPC-C implementation [23]. The simulation (Fig. 3(b)) comprises the following steps: (i) Recording of the raw-trace; (ii-a) Simulation of SIAS and SI resulting in I/O traces; (ii-b) remapping of the traces, creating SI-PL and SI-PG traces; (iii) I/O trace execution on a physical SSD (Intel X25-E SLC) using FIO; and (iv) validation using our SIAS prototype in PostgreSQL, which was installed on the same hardware. We describe all those steps in detail in the following paragraphs.

Instrumentation. A default, out of the box PostgreSQL (9.1.4.) was used to record the trace. It was instrumented utilising TPC-C (DBT2 v0.41)[23] with the PostgreSQL default page size of 8KB. All appends were conducted using this granularity. The used Fedora Linux (kernel 2.6.41) included the systemtap extension (translator 1.6; driver 0.152).

Raw Trace. The raw trace (Fig. 3(b)) contains: (i) tuples and the operations executed on them; (ii) the visibility decision for each tuple; (iii) the mapping of tuples to pages. We record each operation on a tuple and trace the visibility decision for that tuple. By setting probing points accordingly within the transaction- and buffer-manager, we eliminate their influence and are able to simulate the raw I/O of heap-data (non-index data) tuples. The resulting raw-trace is fed to the simulator.



(a) Blocktrace on SSD: SI-V vs. SIAS-LO



(b) Simulation Process

Fig. 3. Blocktraces and Simulation Process

Simulator. SI and SIAS are simulated based on the raw trace including visibility checks and the resulting storage accesses. During the simulation the DB state is re-created according to the respective approach (Fig. 2). Hence the simulated databases always contain exactly the same tuples as the original DB. The only difference is the permutation of the tuples' location; tuples reside on different pages within the simulated DB.

SIAS creates a new tuple mapping: when a tuple is inserted into the original DB (raw trace), the tuple is inserted in its initial version, augmented by the SIAS attributes. The baseline SIAS algorithm (SIAS-L) algorithmically generates a local append, yielding one append region per relation. In order to simulate SIAS-G, an additional mapping is performed, analogous to the page-append LbSM.

As a result of the simulation process block-level traces are generated. These reflect the I/O access pattern that a DMBS would execute against the storage. Subsequently the block-level traces are executed on an a real SSD using FIO, which allows us to precisely investigate the influence of I/O parallelism and raw access.

FIO Trace. The FIO I/O benchmark guarantees controlled trace execution, repeatable results, configurable I/O parallelism and reliable metrics. FIO was configured using the *libaio* library accessing an Intel X25-E SSD via direct I/O as a raw device. The raw device had no filesystem layer in between. We consider the SSD as a black box, which means that no tuning for device-specific properties was applied. To avoid SSD state dependencies we executed a series of 8KB random writes after each single run.

Validation. We implemented the SIAS prototype in PostgreSQL. This prototype was validated under a TPC-C workload. The write patterns generated by our simulation and the PostgreSQL SIAS prototype are the same (see Fig. 3(a)). In terms of I/O parallelism both (PostgreSQL prototype and simulation) achieve: (i) compareable performance; (ii) similar write patterns; and (iii) the same net/gross write overhead reduction.

Load Characterisation. We used the DBT2 benchmark v0.41 [9] which is an open source TPC-C [23] implementation. DBT2 was configured for two traces. Both traces used 10 clients per warehouse and a total of 200 warehouses. *Trace I* with a runtime of 60 minutes and *Trace II* with a runtime of 90 minutes. Based on these two traces we also investigate the impact on space management, chain length etc.

7 Results

I/O Performance and Parallelism. We measured the performance of the algorithms discussed in Section 2 and shown in Fig. 2. We configured FIO with different queue depths (QD) ranging from 1 (no parallelism) to 32 (the maximum queue depth of our Intel X25-E SSD). *I/O performance:* In general, SI-V and SI-NV (SI with and without Vacuum/garbage collection) show the lowest performance for *Trace I* and *Trace II*: the read IOPS of SI-V are the lowest as depicted in Fig. 4a, 4b, 4c and 4d, therefore the runtime of SI-V and SI-NV is significantly higher (Fig. 4e and 4f). Figure 4a and 4c both illustrate the same trace. Figure 4a illustrates the differences between SIAS and SI-P in both global and local implementation variants. Figure 4c additionally displays the general in-place update approach of SI. Furthermore, the I/O performance (seeks, throughput, access distribution) over time for 32 QD is depicted in Fig. 6; SI-V needs more than twice the time of SIAS-L variants. The runtime of the page-append LbSM variants is 2.1x the runtime of SIAS for *Trace I* (Fig. 4d) and *Trace II* (Fig. 4e, 4f).

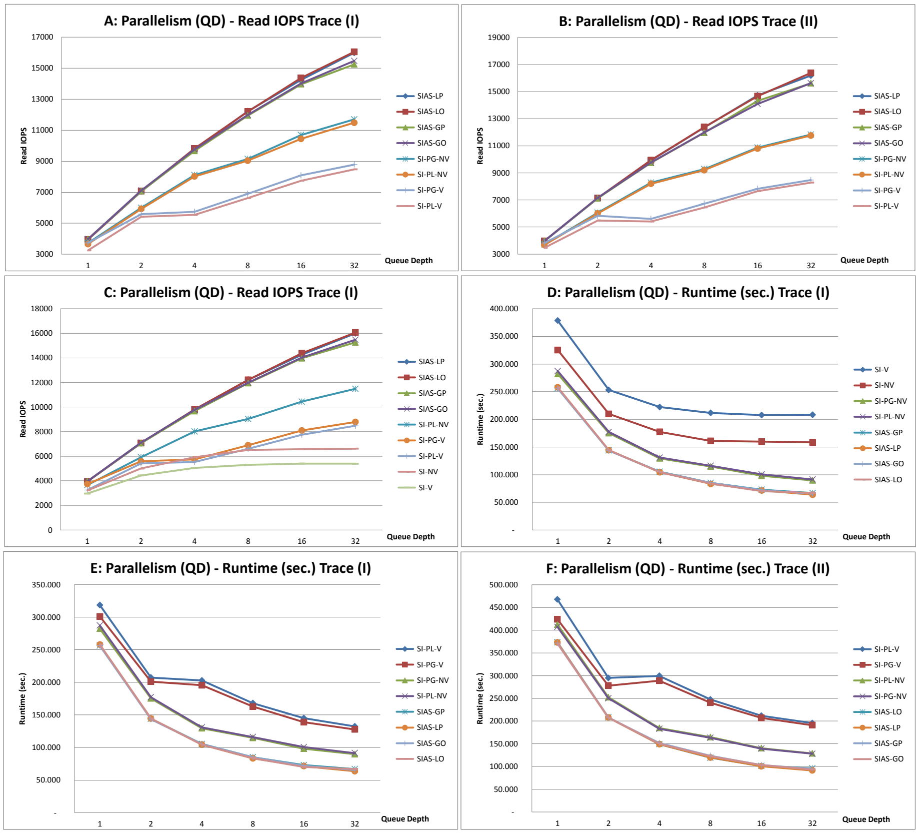


Fig. 4. I/O Parallelism: Read IOPS vs. Queue Depth for *Trace I* (a, c, d, e) and *Trace II* (b, f)

Without parallelism: (i) the SIAS I/O rate is 34% higher than SI-V and 23% higher than SI-NV; (ii) SI-NV is approx. 8% faster than SI-V.

I/O parallelism: Fig. 4 shows that SI-V and SI-NV improved up to a QD of two (2 parallel I/O requests), stagnated at four and reached saturation at a QD of eight; hence no leverage of parallelism. The page-append LbSM variants SI-PG and SI-PL are up to 73% faster than the in-place update SI variants SI-V and SI-NV (QD of 32). Without parallelism SI-PL is 13% faster than SI if garbage collection (Vacuum) is activated (up to 25% higher read IOPS than SI-V if vacuum is deactivated). SI-PL is marginally slower than SI-PG (Fig. 4c and 4d). Since SI-PL has to write at multiple locations, more seeks (random writes) are required than in SI-PG, as illustrated in Fig. 5 (Seek Count) – the append-log region for reads/writes of each relation is visible as a straight line.

With increasing parallelism approaches using one append region per relation have the advantage over single region approaches.

Garbage Collection (Vacuum): all variants with enabled vacuum are significantly slower than their counterparts. This trend is intensified by a higher degree of parallelism. In Fig. 4a and 4b we observe that vacuum creates significant overhead when using the page-append LbSM. Starting with a queue depth of four, page-append LbSM variants loose up to 35% IOPS when using vacuum (Fig. 4a and 4b). SI-PG-NV and SI-PL-NV scale up to the maximum queue depth experiencing a slight stagnation at queue depths larger than four (Fig. 4a and 4b). SI-PG-V and SI-PL-V benefit from higher queue depths but not as much as the variants with deactivated vacuum. Garbage collection mechanisms are therefore not beneficial for page-append LbSMs. SIAS scales almost linearly with increasing parallelism and benefits from a high queue depth. The difference between pessimistic and optimistic SIAS is not significant but enhances with increasing levels of parallelism as depicted overall in Fig. 4. Global and local variants of SIAS perform equally well at lower levels of parallelism. With increasing parallelism the local approach is approx. 5% faster than the global approach, hence making optimal use of the Flash device's parallelism. On *Trace I*, SIAS (in all variants) is up to 2.99x faster than SI-V, 2.43x faster than SI-PL-V/SI-PG-V and approx. 40% faster than SI-PG-NV/SI-PL-NV. Since the performance difference between the global and local implementation of SIAS is marginal and in favour of the local variant, it is not justified to create and maintain an additional page mapping as it is necessary for the global variant (SIAS-G). *Trace II* shows results analogous to *Trace I*. The I/O rate directly correlates with the runtime of the traces. The tendencies observed in this section are confirmed. The in-place approaches SI-V and SI-NV need the most time to complete the trace as depicted in Fig. 4d and Fig.6. SI-PL-NV and SI-PG-NV show almost identical runtime behaviour as well as SI-PL-V and SI-PG-V (Fig. 4e). SIAS is in all four implementations faster than the other approaches (Fig. 4).

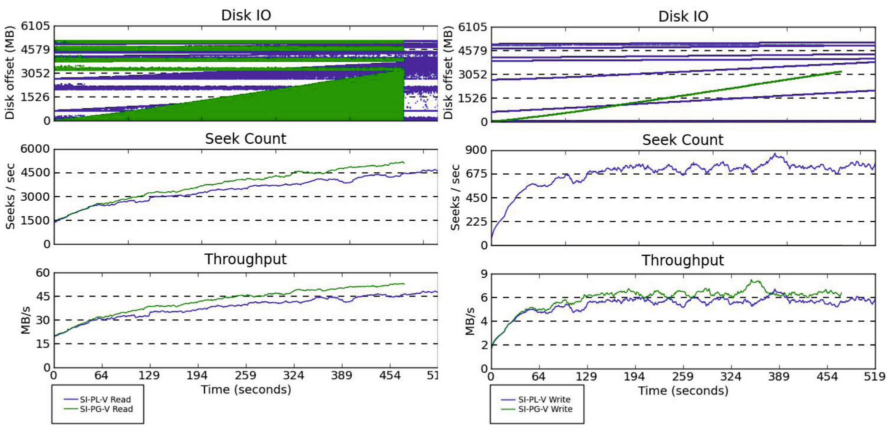


Fig. 5. Read Write Blocktrace on Physical Device: SI-PL vs. SI-PG

Read/Write Overhead. Non-Vacuum SI variants (SI-PG-NV, SI-PL-NV and SI-NV) write 966MB in *Trace I* and 1396MB in *Trace II*. SI variants performing Vacuum (SI-PG-V, SI-PL-V and SI-V) write 1304.6MB in *Trace I* and 1975.3 in *Trace II*. A key feature of SIAS is the significant write reduction of up to *52 times*. SIAS writes (in all variants) 25MB in *Trace I* and 39.9MB in *Trace II*. The write overhead is reduced to a fragment of the usual amount, which is a direct consequence of the out-of-place invalidation, logical tuple appends and dense filling of pages. The metadata update to invalidate a tuple version in SI leads to an update of the page in which this version resides, although the data-load of that version is unchanged. Additionally the new version has to be stored. SIAS avoids such metadata updates. Pages are packed more dense and tuple versions of subsequent access are most likely cached.

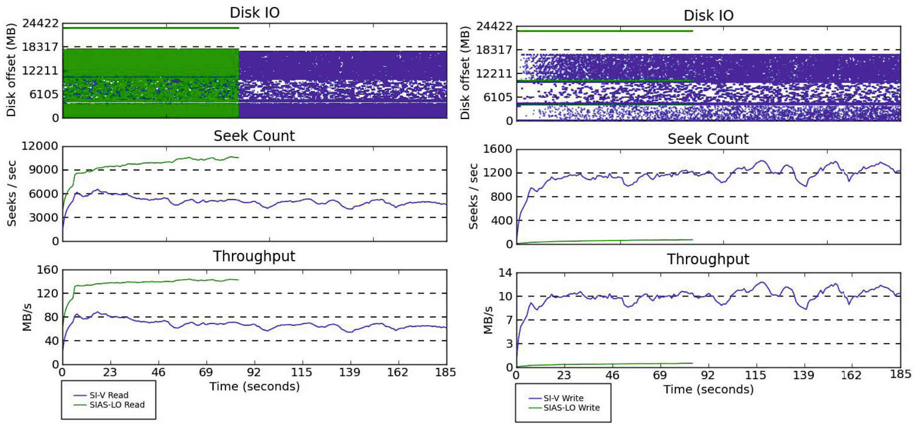


Fig. 6. Read Write Blocktrace on Physical Device: SI-V vs. SIAS-LO

8 Conclusion

We compared in-place storage management and page-/tuple-based LbSM approaches in conjunction with multi versioning databases on new storage technologies and elaborated the influence of one single or multiple append regions. Our findings show that while page-append LbSM approaches are better suitable for new storage technologies, they can be optimised by implementing tuple-based LbSM directly into the MV-DBMS. We implemented SIAS, a tuple-append LbSM within a MV-DBMS which algorithmically generates local append behaviour. SIAS leverages the properties of Flash storage, achieves high performance, scales almost linearly with growing parallelism and exhibits a significant write reduction. Our experimens show that: a) traditional LbSM approaches are up to 73% faster than their in-place update counterparts; b) SIAS tuple-version granularity append is up to 2.99x faster (IOPS and runtime) than in-place update approaches; c) SIAS reduces the write overhead up to 52 times; d) in SIAS using exclusive append regions per relation is up to 5% faster than using one append region for all relations.

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

References

1. Agrawal, N., Prabhakaran, V., et al.: Design tradeoffs for ssd performance. In: Proc. ATC 2008, pp. 57–70 (2008)
2. Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., O'Neil, P.: A critique of ansi sql isolation levels. In: Proc. SIGMOD 1995, pp. 1–10 (1995)
3. Bernstein, P.A., Reid, C.W., Das, S.: Hyder - a transactional record manager for shared flash. In: CIDR 2011, pp. 9–20 (2011)
4. Bober, P., Carey, M.: On mixing queries and transactions via multiversion locking. In: Proc. IEEE CS Intl. Conf. No. 8 on Data Engineering, Tempe, AZ (February 1992)
5. Cahill, M.J., Röhm, U., Fekete, A.D.: Serializable isolation for snapshot databases. In: Proc. SIGMOD 2008, pp. 729–738 (2008)
6. Carey, M.J., Muhanna, W.A.: The performance of multiversion concurrency control algorithms. ACM Trans. on Computer Sys. 4(4), 338 (1986)
7. Chan, A., Fox, S., Lin, W.-T.K., Nori, A., Ries, D.R.: The implementation of an integrated concurrency control and recovery scheme. In: Proc. SIGMOD 1982 (June 1982)
8. Chen, F., Koufaty, D.A., Zhang, X.: Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In: Proc. SIGMETRICS 2009 (2009)
9. Database Test Suite DBT2, `http://osdldbt.sourceforge.net`
10. Gottstein, R., Petrov, I., Buchmann, A.: SI-CV: Snapshot isolation with co-located versions. In: Nambiar, R., Poess, M. (eds.) TPCTC 2011. LNCS, vol. 7144, pp. 123–136. Springer, Heidelberg (2012)
11. Gottstein, R., Petrov, I., Buchmann, A.: Aspects of append-based database storage management on flash memories. In: Proc. of DBKDA 2013, pp. 116–120. IARIA (2013)
12. Grund, M., Krüger, J., Plattner, H., Zeier, A., Cudre-Mauroux, P., Madden, S.: Hyrise: a main memory hybrid storage engine. Proc. VLDB Endow. 4(2), 105–116 (2010)
13. Kemper, A., Neumann, T.: Hyper: A hybrid oltp and olap main memory database system based on virtual memory snapshots. In: ICDE (2011)
14. Krueger, J., Kim, C., Grund, M., Satish, N., Schwalb, D., Chhugani, J., Plattner, H., Dubey, P., Zeier, A.: Fast updates on read-optimized databases using multi-core cpus. Proc. VLDB Endow. 5(1), 61–72 (2011)
15. Majumdar, D.: A quick survey of multiversion concurrency algorithms
16. O'Neil, P., Cheng, E., Gawlick, D., O'Neil, E.: The log-structured merge-tree (1996)
17. Petrov, I., Gottstein, R., Ivanov, T., Bausch, D., Buchmann, A.P.: Page size selection for OLTP databases on SSD storage. JIDM 2(1), 11–18 (2011)
18. R. Gottstein, I. Petrov and A. Buchmann. SIAS: On Linking Multiple Tuple Versions in Append DBMS (submitted)
19. Revilak, S., O'Neil, P., O'Neil, E.: Precisely serializable snapshot isolation (pssi). In: Proc. ICDE 2011, pp. 482–493 (2011)
20. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database Systems Concepts, 4th edn. McGraw-Hill Higher Education (2001)
21. Stoica, R., Athanassoulis, M., Johnson, R., Ailamaki, A.: Evaluating and repairing write performance on flash devices. In: Boncz, P.A., Ross, K.A. (eds.) Proc. DaMoN 2009, pp. 9–14 (2009)
22. Stonebraker, M., Rowe, L.A., Hirohama, M.: The implementation of postgres. IEEE Trans. on Knowledge and Data Eng. 2(1), 125 (1990)
23. TPC Benchmark C Standard Specification, `http://www.tpc.org/tpcc/spec/tpcc_current.pdf`
24. Wu, S., Kemme, B.: Postgres-r(si): Combining replica control with concurrency control based on snapshot isolation. In: Proc. ICDE 2005, pp. 422–433 (2005)