IPLization of PostgreSQL on NVDIMM

Joobo Shim, Sang-Won Lee
School of Information & Communications Engr.
Sungkyunkwan University
Suwon 440-746, Korea
joobo95@skku.edu, swlee@skku.edu

***Abstract*—A novel approach to DBMS design called In-Page Logging (IPL) was proposed a decade ago. The IPL system exploits characteristic of flash memory; asymmetric write/read speed. This approach manages per page log in erase unit of flash memory and avoids page write but saves redo log. When the page is required merge operation, which is instant recovery process, generates updated version of page using old page and its redo log. In fact this never has been implemented to real DBMS system since lack of fast, persistent, byte-addressable, and affordable device. Since NVDIMM matches with concept of IPL, we implemented IPL to PostgreSQL, a commercial open source DBMS, employing it as IPL log device. The experiment showed improvement of reducing write amount, which leads to performance gain and SSD lifetime increase.**

Keywords: In-page logging, PostgreSQL, NVDIMM

I. INTRODUCTION

SSDs are attracting attention because they are smaller in size, lighter in weight, stronger against shock, lower electricity consumption, less noise, and faster read/write performance than conventional HDDs. Currently, many SSDs are used in personal computers as well as data servers. With the development of technology, prices are dropping gradually, performance and capacity are getting better, and it is expected to be the most widely used storage replacing HDD. SSDs have several unique features. The NAND flash cell can only be used for a predetermined erase-program cycle, and has a minimum writing unit that is increased from SLC, MLC to TLC. These characteristics occur disadvantages in aspect of wearing and performance in environments where small random writes occur frequently, such as OLTP. Write amplification phenomenon, which changes write of few bytes to the write of several pages, also occurs, worsening the problem [1].

SSD also has asymmetric read/write speeds, which read is significantly faster than write. A decade ago, a novel approach to DBMS design called IPL was proposed [2]. The IPL system overcomes the difficulties of exploiting benefits of flash memory. Write/erase operations caused by small random writes have high latency. In order to avoid this, only per-page logs are stored in the storage, not the whole page, and eventually merged to database. When this method was introduced, it was promising but never applied to real DBMS due to the lack of memory device which is fast, persistent and affordable for byte-addressable small delta writing. Therefore, there have been attempts to apply it to byte-addressable NVRAM such as PCRAM [3].

Adopting NVDIMM makes it possible to apply the new logging system previously. NVDIMMs have speed of DRAM, are persistent, and are cheaper than other NVRAMs. In fact, NVDIMM interface is supported by Linux, MS Windows, and hardware servers like HP ProLiant. It is the most realistic NVRAM solution and the most suitable storage for IPL concept.

The composition of this paper is as follows. Chapter 2 explains PostgreSQL tuple management system to help understanding following section. Section 3 introduces how IPLization of PostgreSQL was done as well as IPL approach. Section 4 analyzes the performance evaluation environment and results of the comparing original and modified system. Section 5 briefly present related works. We conclude the paper presenting future research.

II. POSTGRESQL TUPLE MANAGEMENT

A. *PostgreSQL Page And Tuple Structure*

PostgreSQL manages user-created data in the form of heap data. Page contains item pointer and actual tuple, and tuple is accessible through item pointer. Each tuple represents a row of relations. Tuple has value called tmax and tmin which are given at creation and deletion. When performing an insert, a new tuple is created on a empty space in page. When a delete is performed, only the delete mark is set, and the tuple data is not deleted. Update operation is equivalent to performing delete and insert once each. To take care of continuously growing size of heap data, PostgreSQL frees the space of deleted tuples through vacuum operation. Fig .1 is the result of performing three DML queries to an empty page. TID states for transaction ID.

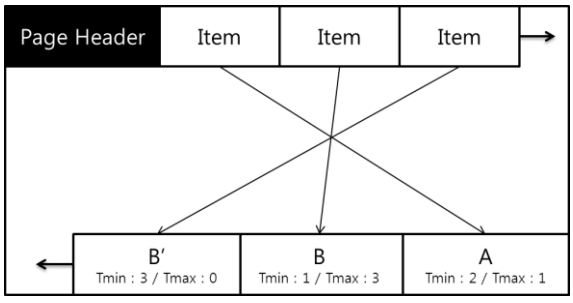

Figure 1. Result of executing 3 transactions.

(1) *TID1* : Insert A, B
(2) *TID2* : Delete A
(3) *TID3* : Update B to B'

Executing (1) creates tuple A and B and assigns tmin value 1. Executing (2) assigns tmax value 2 to tuple A. Executing (3) assigns tmax value 3 to tuple B, creates tuple B' and assigns tmin value 3.

B. MVCC

As example above, tmin and tmax are tid values that are given when creating and deleting tuples. Tmax value serves as delete mark. PostgreSQL handles MVCC using these values. When a transaction scans heap data, only the tuple with visibility is accessible. Visibility is obtained when the tid of the transaction is greater than tmin and less than tmax. That is, tmin is the minimum tid value to get visibility, and tmax is the maximum tid value for it. PostgreSQL keeps commit log separately and stores the status of transactions. The tuple of aborted transactions cannot have visibility because it is marked abort in commit log.

C. WAL Logging System

PostgreSQL's initial design architecture suggests a very simple logging system, but currently does not use it because of durability issues. The process of WAL logging in PostgreSQL is shown in Fig. 2. When a client requests data manipulate language (DML) query the server creates WAL log for each operation. PostgreSQL logs every operation that modifies page data.

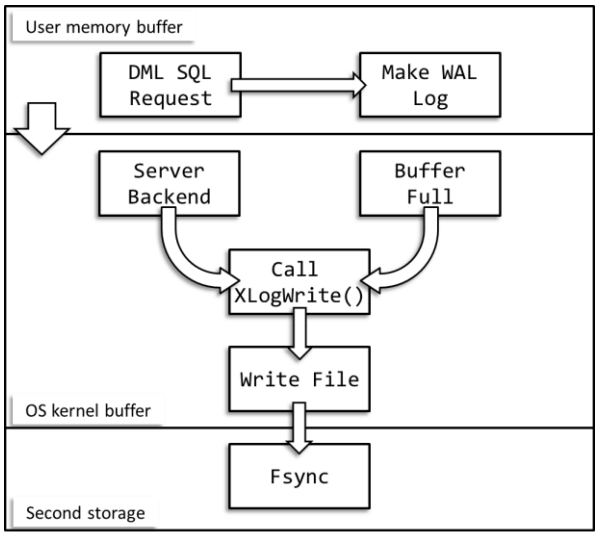

Figure 2. PostgreSQL WAL Log System Architecture

Since the created log cannot be recovered if system crash occurs while the server is processing the request. It tries to save the log for every log creation. Log file is switched by backend of server when the timeout set by the DBMS occurs or certain number of transactions is performed or when there is no more log capacity on log buffer. Logging is performed by XLogWrite function, which opens a file and saves created log and calls fsync. WAL logs are stored in the order of the LSN, and the LSN value indicates the location of the actual file.

When the WAL log is created by the SQL statement, the log contents are stored in the user memory buffer as program internal variable. When the WAL log is saved to a file, it is written to the OS kernel buffer and stored in the durable memory when fsync is called. The write and fsync functions used to access the shared data are synchronous and require an exclusive lock. Because the data processing speed of the storage device is considerably slower than the CPU, all transactions that do not have locks must wait, and the entire database system experience bottleneck due to this latency.

The proportion of WAL logs generated by TPCC is in the order of update(28%), btree insert(26%), tuple level lock(17%), insert(14%), heap clean(12%), delete(1%), others(1%).

D. Recovery

PostgreSQL creates one WAL log for each tuple change, and manages it in the order of the LSN. If a system crash occurs during transaction processing, the committed data and stored data pages are not the same. For durability, DBMS uses redo log to recover data. The data before checkpoint is stored in the storage, but the contents of the buffer pool are not recovered yet. Recovery process reads WAL log starting from checkpoint. If LSN value of the WAL log is larger than LSN value of the page, it means that the newly applied data are not reflected. Accordingly, the page is updated through the redo function.

As described in PostgreSQL's MVCC, all data changes are made in an increasing direction, so if only redo process is repeated, not only the insert, but also the delete and update operations will be automatically reflected. Since there is no need for rollback, PostgreSQL can complete the recovery with only redo logic similar to the DML operation. The WAL log has pages, data offsets, operation information, and applied tuple values. The DBMS data after reflecting WAL log is guaranteed to contain all the committed data of the moment of crash.

As a result, if a system crash occurs, all that PostgreSQL has done is applying logs sequentially. This nature simplifies the merge operation of IPL approach for PostgreSQL.

III. IPLIZATION OF POSTGRESQL ON NVDIMM

A. In Page Logging Approach

In-Page Logging is a novel design for flash-based DBMS. It can overcome limitations of SSD and exploit its advantages. IPL manages per page log in erase unit of flash memory and shows difference from existing system in 3 situations. The first is when writing to disk. At that time, IPL does not write data page but only records log data. Then the old page will remain on disk. Second is when reading from disk. It combines old version of the page with the log and perform instant recovery to generate updated page, which is merge operation. Since merge is fast enough, the host will only see new page from disk. Last, when log sector is full.

When the allocated log area is full, logs and old pages are merged to updated version and written to disk.

This allows the DBMS to replace page writes to log writes, which is smaller, by merging once every time the log area is full. Instead of avoiding one write operation, one log read and one instant recovery are included. However, SSDs are much slower in writes than read, even if a merge process is added, this obviously causes performance gain. In addition, because the write amount to disk is reduced, lifespan SSD is expected to increase.

Unfortunately, flash-based IPL has never been implemented in real DBMS. The reason is that the current SSD only supports page wise writing, and cannot effectively write few bytes to the IPL log area. There have been attempts to apply it using byte-addressable NVRAM such as PCRAM, but the price of such device is too expensive to be commercialized. On the other hand, NVDIMM is as fast as DRAM, persistent and cheap. Applying IPL to DBMS by adopting NVDIMM as IPL log area will improve SSD storage performance and lifetime.

B. IPLized PostgreSQL

The IPL log area was allocated on NVDIMM and managed in page wise manner. PostgreSQL manages the log by LSN. Since IPL approach needs to manage logs in page units, we added a procedure to capture WALs log and organize it per page units. We implemented the three situations described above. First, when writing to disk, we blocked write operation occurred in buffer pool or caused by the background flusher. Second when reading from disk, the old page merges with corresponding IPL logs and passes updated version to the host. The merge process is implemented by benchmarking recovery logic of PostgreSQL. Lastly, when log sector is full, it does not prevent writing. It works same as the existing process, and cleans the log area. Except for new implementation of merge logic for IPL, there are only tens of lines changed in existing code.

Since operations such as btree split covers several pages and require too much time for merge operation, we have defined target pages, operations suitable for IPL. Heap file and index file are IPLized and IPL is applied to six operations: insert, delete, update, tuple level lock, heap clean, and btree insert. The heap clean operation here refers to the page wise vacuum that PostgreSQL performs for every page read. When the IPL is not performed, the page is operated just like the existing DBMS.

Some operations on the DBMS occasionally generate a single WAL log but take place across multiple pages. Suppose tuple A was updated which is one delete and one insert in PostgreSQL. Old version remains on page 100 and new version was created on page 110. If so, page 100 will have a delete IPL Log, and 110 will have an insert IPL Log. When each page is requested by DBMS, the merge process is performed without regard to each other. Page 100 and Page 110 are independent from the viewpoint of IPL merge operation, although both have been changed by one operation. It is possible because updated version and merged version are idempotent pages.

Fig. 3 illustrates how existing DBMS and IPL approach work in PostgreSQL. The time is specified for file I/O. The solid line indicates the IPL approach and dotted line indicates the existing DBMS. As it can be seen, IPL replaces one write with one read and merge, which is much faster.

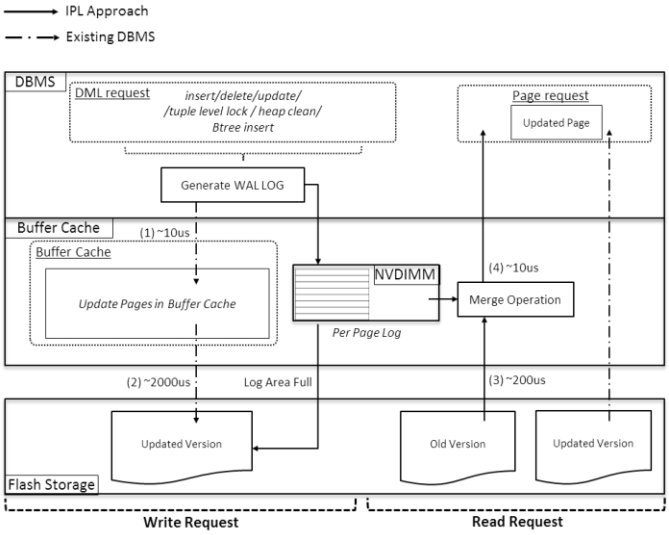

Figure 3. Comparison of Existing DBMS and IPL Approach

IV. PERFORMANCE EVALUATION

TABLE1. EXPERIMENTAL ENVIRONMENT DETAILS

OS	Linux (Kernel 3.13.0-74)
CPU	Intel® Core™ i5-2500k CPU @ 3.30GHz (4 CPUs)
DRAM	6 GB
STORAGE	Samsung 840 PRO SSD 256GB
NVDIMM	Emulated NVDIMM 2GB
DBMS	PostgreSQL 9.4.5
BENCHMARK	BenchmarkSQL 5.0

In performance evaluation, we used PostgreSQL 9.4.5 for DBMS, and BenchmarkSQL 5.0 for the benchmark tool. Experimental environment details are in Table.1. In TPC-C benchmarking setup, the value of warehouse is 100 (about 11 GB), and user is 32, run time is 20 minutes. NVDIMM was emulated using Linux PMEM interface at DRAM.

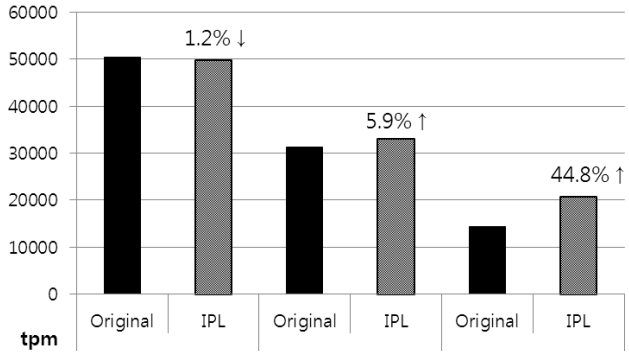

Figure 4. Tpm Analysis For Each File I/O configuration.

Fig.4 shows the transactional performance differences between original and IPLized PostgreSQL in each file I/O configuration. In normal file system, IPLized DBMS performance has dropped by 1.2%, in direct I/O environment increased by 5.9%, and in osync environment increased by 44.8%. For a normal file system, PostgreSQL only supports buffered I/O mode, which does IO operations against buffer cache. Therefore, the effort to reduce the amount of write toward storage was not effective.

Also, the process of storing IPL log and merge process speed became similar to read/write speed, which generate minor overhead. The IPL approach shows better performance at situation where large write toward storage occurs. To demonstrate the maximum performance improvement of IPL approach, following experiments were conducted using the sync option.

TABLE II. TPM, READ, WRITE ANALYSIS AT EACH LOG SIZE.

LogSize	0	256	512	1024	2048
tpmtotal	8267.02	9935.67	12530.38	13402.00	14372.38
WR/10000TX	29412.75	8478.18	6550.08	5630.43	4863.33
RD/10000TX	8409.13	28847.31	28166.48	31259.59	32824.70

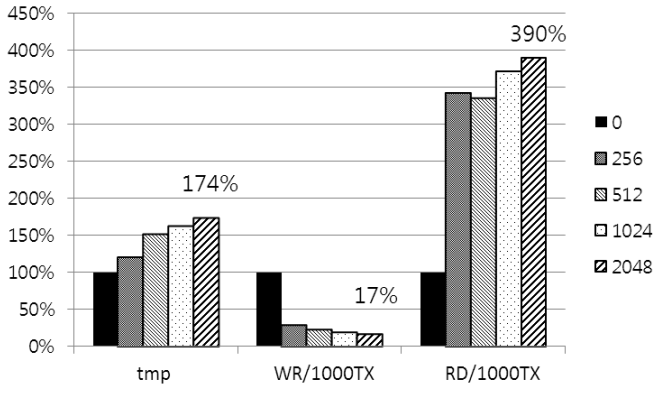

Figure 5. Tpm, Read, Write Analysis At Each Log Size.

Table.2 and Fig.5 show tmp, write/read amount toward storage according to log size. As the log size increases, the amount of write is greatly reduced and read is increased. It is the characteristic of IPL approach that write is replaced by read and merge operation. As a result, the transaction throughput has improved by up to 74%. The amount of write has been reduced by up to 17%, which would help expanding life of the SSD.

IPLized PostgreSQL showed better performance at write mount and transaction throughput. The amount of read has increased, but by the characteristics of asymmetric read/write speed on SSD, it did not cause performance degradation.

V. RELATED WORKS

Reducing write amount of SSD based DBMS is not only good in terms of performance, but also extends lifespan of the storage device. Since the size of the log is smaller than the size of the page itself in most cases, storing log instead of page in persistent storage device can reduce write amount. As computing devices such as CPU and DRAM are getting better, performing instant recovery using old version page and log merely affect performance. The research that employs log-write and instant recovery is adopted by [2] which provides the core idea of the paper. IPL is a novel design of flash-based DBMS as described above and [4] is a transactional DBMS design that provides MVCC and recovery solution using IPL approach. The merge process used in page read can replace the recovery process, thus enabling immediate recovery.

[1] uses a portion of the SSD as a byte-addressable delta-record area using the characteristics of the flash cell and presents a new page-format. Here, instead of using the updated version of the page, old version and delta-record are combined to create updated version. [5] scopes update propagation strategies and give variation to page-based propagation. They present log-based propagation technique using partially ordered log and old page for effective management of log accumulation.

After the emergence of fast and byte addressable NVDIMMs, researches are underway to reduce write amount toward SSDs. [3] is a study applying per-page logging method to SQLite by adopting PCRAM. They effectively solved the write amplification problem, which is characteristic of SQLite, and reduces write amount. [6] is a study that replaced the log area of commercial open source DBMS with NVDIMM. [7] replaces the WAL log area with NVRAM, and suggests an optimized WAL logging system.

VI. CONCLUSION

On this paper, IPL was successfully implemented on commercial open source DBMS adopting NVDIMM. The IPL log area was allocated on NVDIMM and managed in page wise manner. We implemented the three situations. First, when writing to disk, we blocked write operation occurred in buffer pool or caused by the background flusher. Second when reading from disk, the old page merges with corresponding IPL logs and passes updated version to the host. Lastly, when log sector is full it works same as the existing process, and cleans the log area.

We also have defined target pages, operations suitable for IPL. The IPL approach replaces one write with one read and merge, which is much faster. The experiment showed improvement of reducing write amount in proportion to occurrence of write operation on DBMS. The bigger the log size, the smaller the amount of writes. As a result is led to performance gain and SSD lifetime increase.

For the future research, we planned to optimize usage of IPL log space on NVDIMM. Appropriate replacement policy and new features to IPL Log management module will be added. Upgrading IPLized Postgres to transactional IPLized Postgres is also on wish list.

ACKNOWLEDGMENT

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the "SW Starlab" (IITP-2015-0-00314) supervised by the IITP(Institute for Information & communications Technology Promotion)

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2017R1D1A1B03028426)

REFERENCES

[1] H, Sergey, I. Petrov, R, Gosttstein, A. Buchmann. From in-place updates to in-place appends: Revisiting out-of-place updates on flash. In: Proceedings of the 2017 ACM International Conference on Management of Data. ACM, 2017. p. 1571-1586.

[2] S.W.Lee, B.Moon, Design of flash-based DBMS: an in-page logging approach, Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM, 2007.

[3] G. OH, S.Kim, S.W.Lee, B.Moon, Sqlite optimization with phase change memory for mobile applications. Proceedings of the VLDB Endowment, 2015, 8.12: 1454-1465.

[4] S.W.Lee, B.Moon, Transactional In-Page Logging for multiversion read consistency and recovery. In: Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011. p. 876-887.

[5] S. Caetano, L. Lersch, G. Graefe, Update Propagation Strategies for High-Performance OLTP. In: East European Conference on Advances in Databases and Information Systems. Springer International Publishing, 2016. p. 152-165.

[6] J.Shim, S.W.Lee, Performance evaluation of PostgreSQL WAL log based on NVDIMM, Proceedings of the Korea Computer Congress, 2016, p.1923~1925

[7] W.H.Kim, J.Kim, W.Baek, B.Nam, Y.Won, NVWAL: exploiting nvram in write-ahead logging. In: ACM SIGPLAN Notices. ACM, 2016. p. 385-398.