Design of Flash-Based DBMS: An In-Page Logging Approach

Sang-Won Lee

School of Info & Comm Eng
Sungkyunkwan University
Suwon, Korea 440-746
wonlee@ece.skku.ac.kr

Bongki Moon

Department of Computer Science
University of Arizona
Tucson, AZ 85721, U.S.A.
bkmoon@cs.arizona.edu

Introduction

- In recent years, NAND flash memory wins over hard disk in mobile storage market
  - PDA, MP3, mobile phone, digital camera, ...
  - Advantages: size, weight, shock resistance, power consumption, noise …
- Now, compete with hard disk in personal computer market
  - 32GB Flash SSD: M-Tron, Samsung, SanDisk
  - Vendors launched new lines of personal computers only with NAND flash memory instead of hard disk
- In near future, full database servers can run on computing platforms with TB-scale Flash SSD second storage
  - C.G. Hwang predicted twofold increase of NAND flash density each year until 2012 [ProcIEEE 2003]
  - Database workload different from multimedia applications

Characteristics of NAND Flash

- **No in-place update**
  - No data item or sector can be updated in place before erasing it first.
  - An erase unit (16KB or 128 KB) is much larger than a sector.
- **No mechanical latency**
  - Flash memory is an electronic device without moving parts
  - Provides uniform random access speed without seek/rotational latency
- **Asymmetric read & write speed**
  - Read speed is typically at least twice faster than write speed
  - Write (and erase) optimization is critical

Magnetic Disk vs NAND Flash

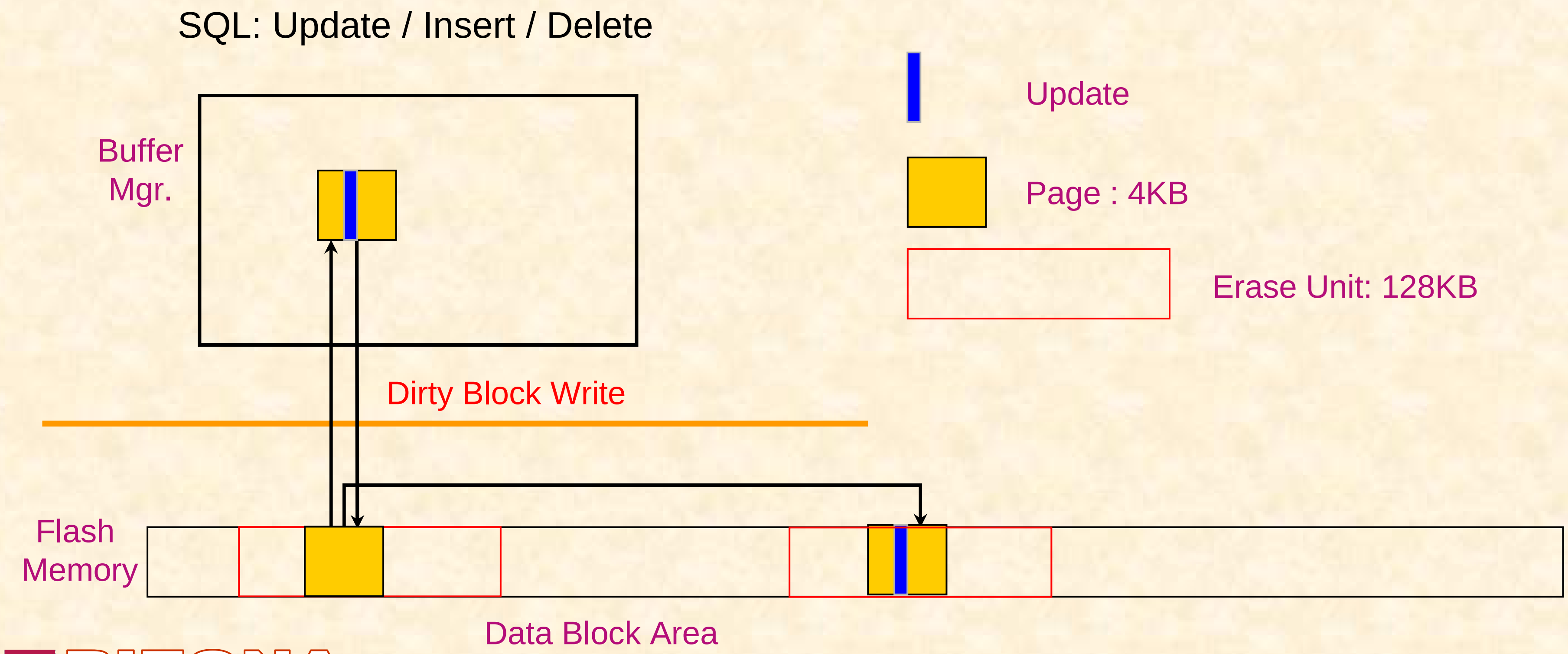
	Read time	Write time	Erase time
Magnetic Disk	**12.7 msec**	**13.7 msec**	**N/A**
NAND Flash	**80 μsec**	**200 μsec**	**1.5 msec**

- **Magnetic Disk : Seagate Barracuda 7200.7 ST380011A**
- **NAND Flash : Samsung K9WAG08U1A 16 Gbits SLC NAND**

- **Unit of read/write: 2KB, Unit of erase: 128KB**

Disk-Based DBMS on Flash Memory

- **What happens if disk-based DBMS runs on NAND Flash?**
  - **Due to No In-place Update, an update causes a write into another clean page**
  - **Consume free sectors quickly causing frequent garbage collection and erase**



Disk-Based DBMS Performance

- **Run SQL queries on a commercial DBMS**
  - **Sequential scan or update of a table**
  - **Non-sequential read or update of a table (via B-tree index)**
- **Experimental settings**
  - **Storage: Magnetic disk vs M-Tron SSD (Samsung flash)**
  - **Data page of 8KB**
  - **10 tuples per page, 640,000 tuples in a table (64,000 pages, 512MB)**

Disk-Based DBMS Performance

- **Read performance :** *The result is not surprising at all*

	Disk	Flash
Sequential	**14.0 sec**	**11.0 sec**
Non-sequential	**61.1 ~ 172.0 sec**	**12.1 ~ 13.1 sec**

- Hard disk
  - Read performance is poor for non-sequential accesses, mainly because of seek and rotational latency
- Flash memory
  - Read performance is insensitive to access patterns

Disk-Based DBMS Performance

- **Write performance**

	Disk	Flash
Sequential	**34.0 sec**	**26.0 sec**
Non-sequential	**151.9 ~ 340.7 sec**	**61.8 ~ 369.9 sec**

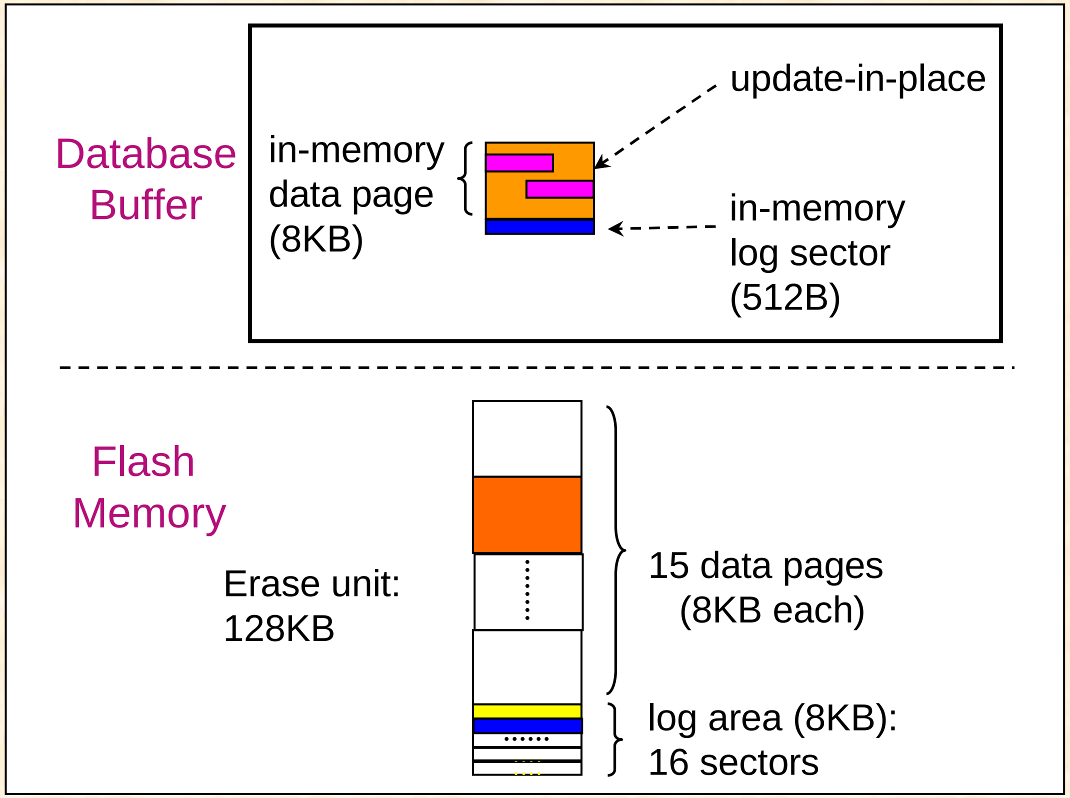
- Hard disk
  - Write performance is poor for non-sequential accesses, mainly because of seek and rotational latency
- Flash memory
  - Write performance is poor (worse than disk) for non-sequential accesses due to out-of-place update and erase operations
  - Demonstrate the ***need of write optimization*** for DBMS running on Flash

In-Page Logging (IPL) Approach

- **Design Principles**
  - **Take advantage of the characteristics of flash memory**
    - **Uniform random access speed**
    - **Fast read speed**
  - **Overcome the "erase-before-write" limitation of flash memory**
  - **Minimize the changes made to the overall DBMS architecture**
    - **Limited to buffer manager and storage manager**
- **Key Ideas**
  - **Changes written to *log* instead of updating them in place**
    - **Avoid frequent write and erase operations**
  - **Log records are *co-located* with data pages**
    - **No need to write them sequentially to a separate log region**
    - **Read current data more efficiently than sequential logging**

Design of the IPL

- **Logging on Per-Page basis in both Memory and Flash**

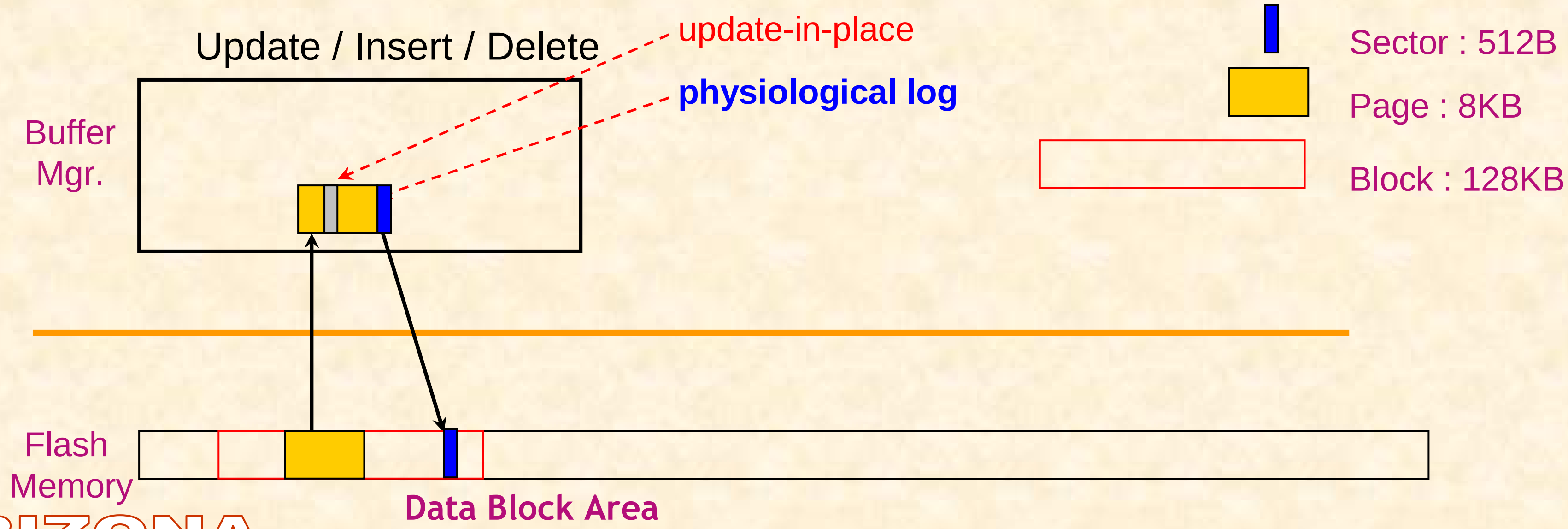


The log area is shared by all the data pages in an erase unit

- An ***In-memory log sector*** can be associated with a buffer frame in memory
  - Allocated on demand when a page becomes dirty
- An ***In-flash log segment*** is allocated in each erase unit

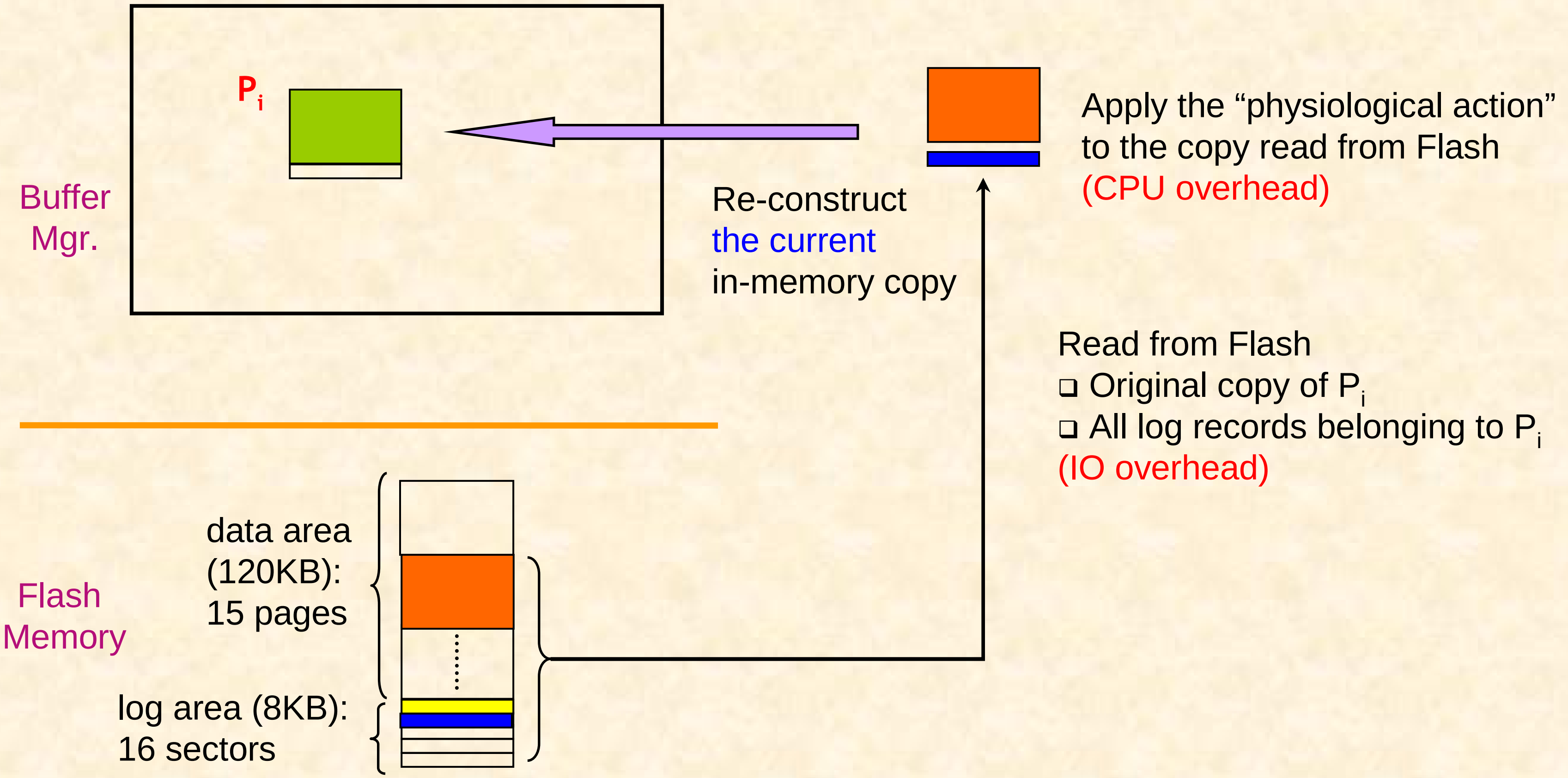
IPL Write

- Data pages in memory
  - Updated in place, and
  - Physiological log records written to its in-memory log sector
- In-memory log sector is written to the in-flash log segment, when
  - Data page is evicted from the buffer pool, or
  - The log sector becomes full
- When a dirty page is evicted, the content is ***not written*** to flash memory
  - The previous version remains intact
- Data pages and their log records are physically co-located in the same erase unit



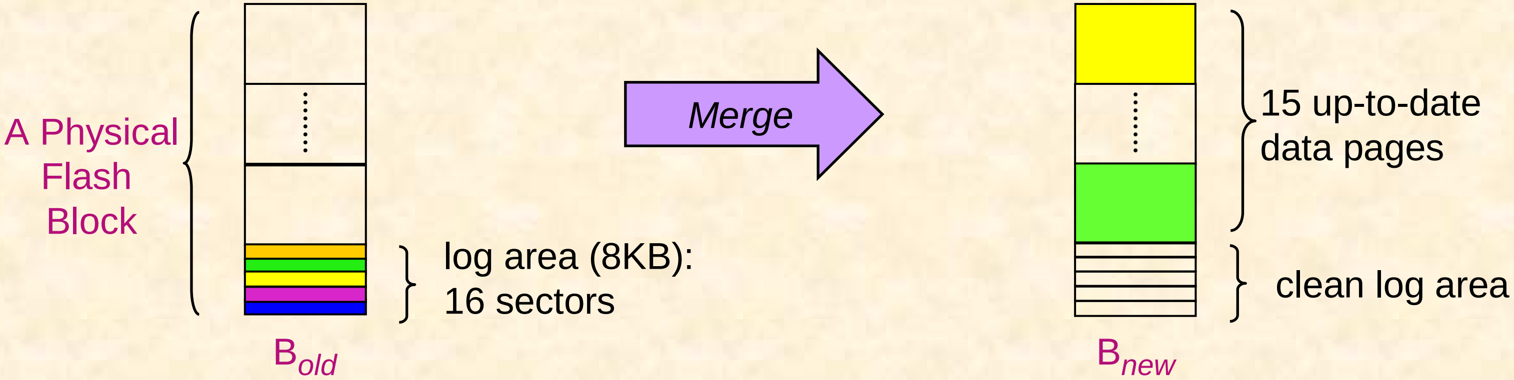
IPL Read

- When a page is read from flash, the current version is computed on the fly



IPL Merge

- **When all free log sectors in an erase unit are consumed**
  - **Log records are applied to the corresponding data pages**
  - **The current data pages are copied into a new erase unit**



IPL Simulation with TPC-C

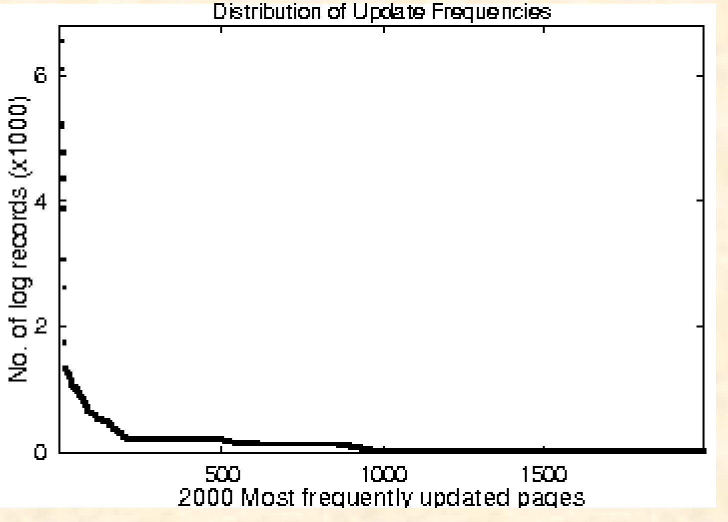
- TPC-C Log Data Generation
  - Run a commercial DBMS to generate reference streams of TPC-C benchmark
    - HammerOra utility used for TPC-C workload generation
  - Each trace contains log records of physiological updates as well as physical page writes
  - Average length of a log record: 20 ~ 50B
- TPC-C Traces
  - 100M.20M.10u: 100MB DB, 20 MB buffer, 10 simulated users
  - 1G.20M.100u: 1GB DB, 20 MB buffer, 100 simulated users
  - 1G.40M.100u: 1GB DB, 40 MB buffer, 100 simulated users

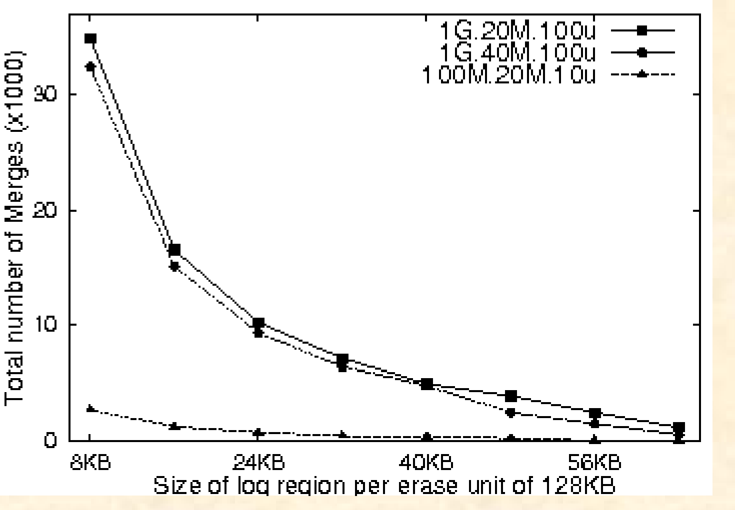
IPL Simulation

- IPL Event-driven Simulator
  - Event-driven simulation of IPL using the TPC-C traces
    - Events: insert/delete/update log, physical writes of data pages
  - For each physiological log,
    - Add the log record to the in-memory log sector; Generate a sector write event if the log sector is full
  - For each physical page write
    - Generate a sector write event; clear the in-memory log sector
  - For each sector write event
    - Increment the write counter
    - If in-flash log segment is full, increment the merge counter
- Parameter setting for the simulator to estimate write performance
  - Write (2KB): 200 us
  - Merge (128KB): 20 ms

Log Segment Size vs Merges

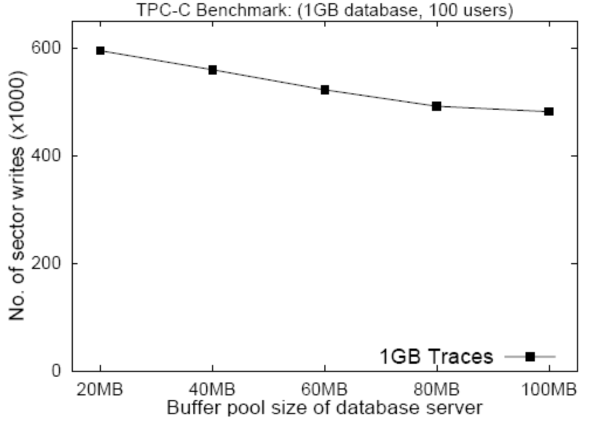
- TPC-C *Write* frequencies are highly skewed (and low temporal locality)
- Erase units containing hot pages consume log sectors quickly
  - Could cause a large number of erase operations
  - More storage but less frequent merges with more log sectors



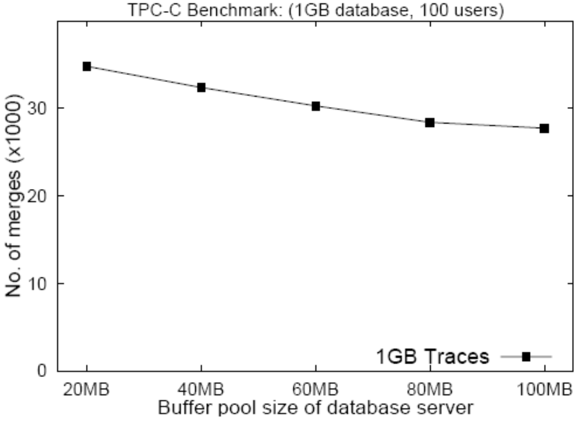


Estimated Write Performance

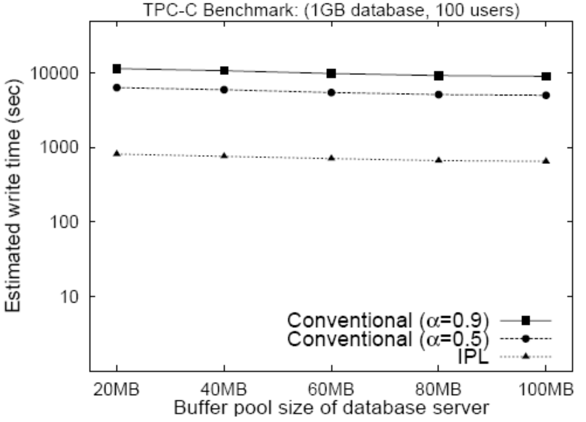
- Performance trend with varying buffer sizes
  - The size of log segment was fixed at 8KB
- Estimated write time
  - With IPL = (# of sector writes) × 200us + (# of merges) × 20ms
  - Without IPL = α × (# of page writes) × 20ms
    - α is the probability that a page write causes the container erase unit to be copied and erased



(a) Total number of sectors written



(b) Total number of merges performed



(c) Estimated write time

Support for Recovery

- IPL helps realize a lean recovery mechanism
  - Additional logging: transaction log and list of dirty pages
- Transaction Commit
  - Similarly to flushing log tail
  - An in-memory log sector is forced out to flash if it contains at least one log record of a committing transaction
  - No explicit REDO action required at system restart
- Transaction Abort
  - De-apply the log records of an aborting transaction
  - Use selective merge instead of regular merge, because it's irreversible
    - If committed, merge the log record
    - If aborted, discard the log record
    - If active, carry over the log record to a new erase unit
  - To avoid a thrashing behavior, allow an erase unit to have overflow log sectors
  - No explicit UNDO action required

Conclusion

- Clear and present evidence that Flash can co-exist or even replace Disk
- IPL approach demonstrates its potential for TPC-C type database applications by
  - Overcoming the “erase-before-write” limitation
  - Exploiting the fast and uniform random access
- IPL also helps realize a lean recovery mechanism