Are You Sure You Want to Use MMAP in Your Database Management System?

MMAP = ◡̈

Junpeng Zhu

Greenplum, VMware, Inc.

zjunpeng@vmware.com

2022 of Greenplum TechTalk
May 3, 2022

GREENPLUM DATABASE®

Presentation Overview

Authors

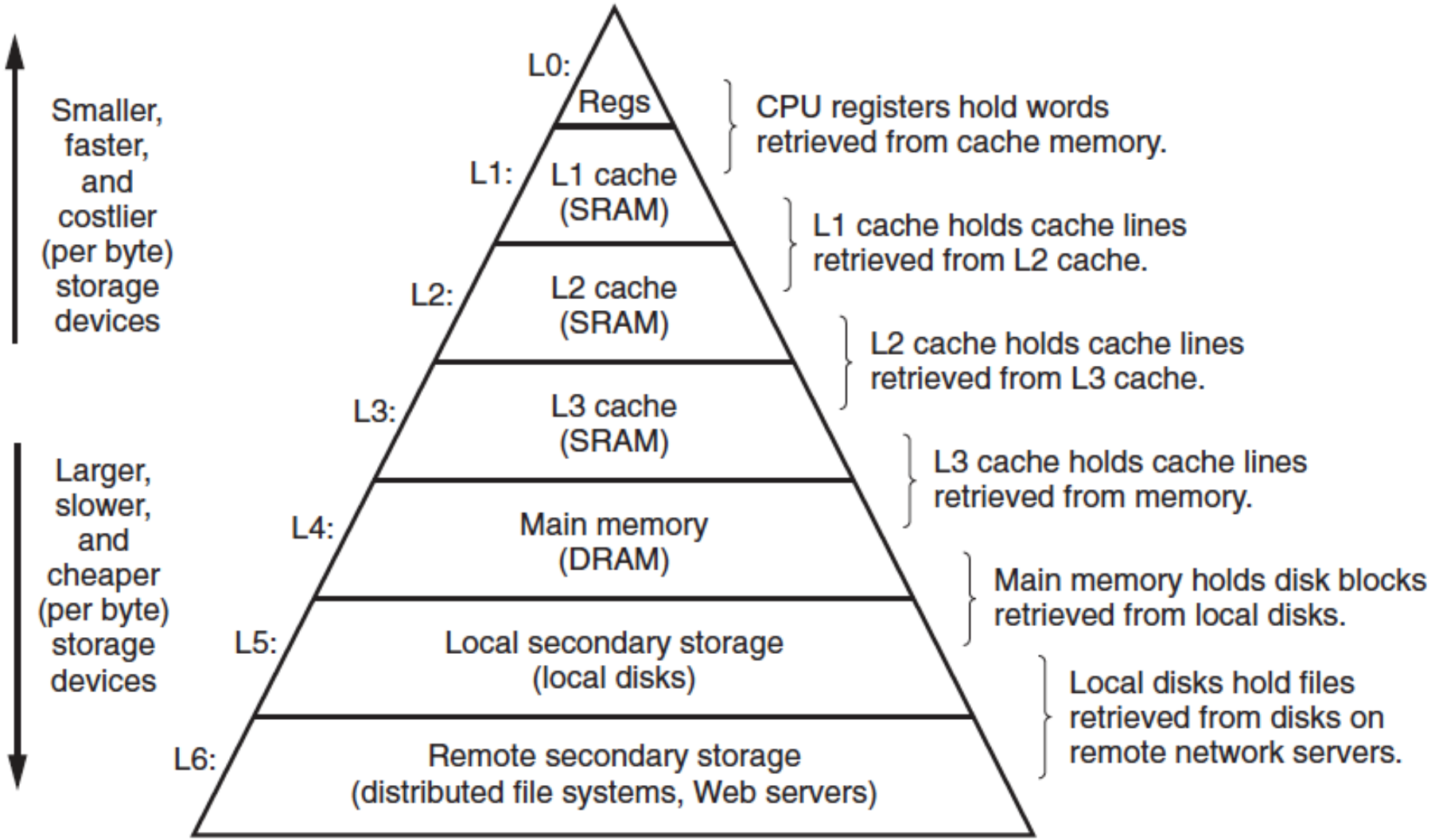
Andrew Crotty
Carnegie Mellon University

Viktor Leis
Friedrich-Alexander-Universität

Andy Pavlo
Carnegie Mellon University

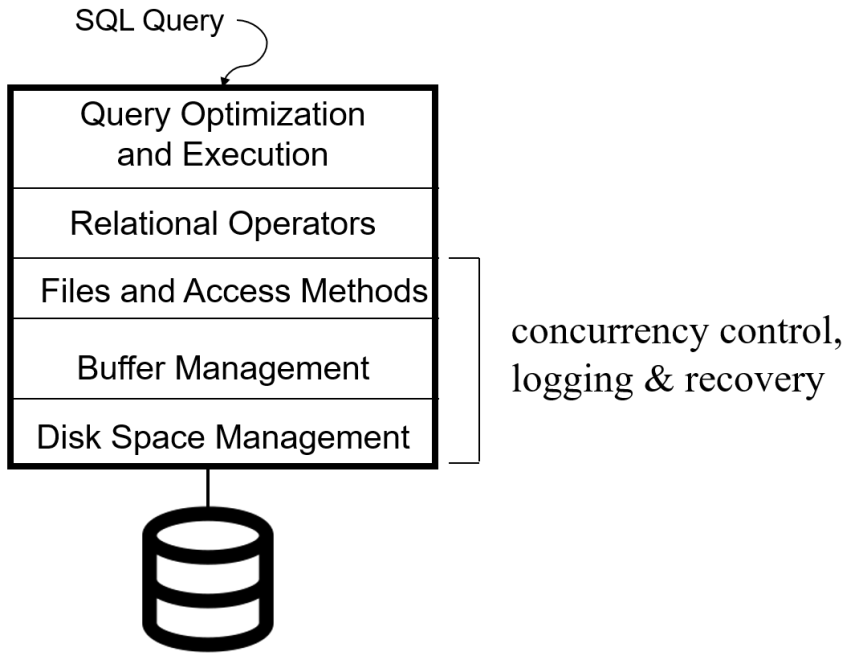
- Andrew Crotty
  - ✓ Andrew Crotty's Bio
  - ✓ Ph.D at Brown, in 2019. Post-doctoral at CMU.
- Viktor Leis
  - ✓ Viktor Leis Bio
  - ✓ Ph.D at TUM, full professor at Friedrich Schiller University Jena.
- Andy Pavlo
  - ✓ Andy Pavlo Bio
  - ✓ Ph.D at Brown, associate professor at CMU.
  - ✓ Never use mmap in a DBMS at his tombstone.

Storage hierarchy, Cont.



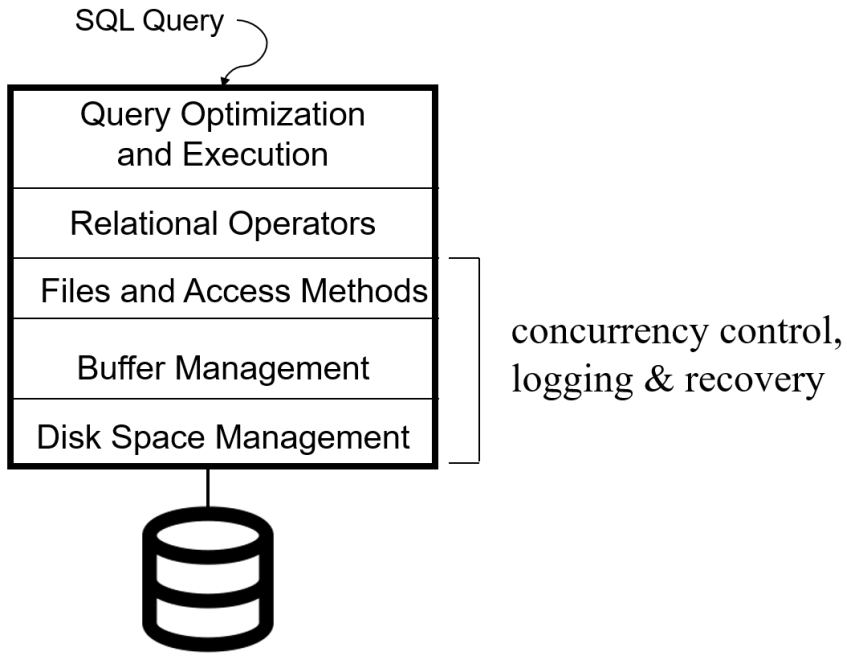
Architecture of RDBMS, Cont.

- Query optimization and execution
- Relational operators
- Files and access methods
- Buffer pool management
- Disk space management



Architecture of RDBMS, Cont.

- Query optimization and execution
- Relational operators
- Files and access methods
- Buffer pool management
- Disk space management



- Crash recovery is awfully difficult!
  - The recovery system depends on behavior of many other components of DBMS, such as concurrency control, buffer management, disk management, and query processing.

Buffer Pool Management, Cont.

- Force policy – make sure that every update is on the DB disk before commit.
  - ✓ Provides durability without REDO logging.
  - ✓ But, can cause poor performance due to a large random write operations.

Buffer Pool Management, Cont.

- Force policy – make sure that every update is on the DB disk before commit.
  - ✓ Provides durability without REDO logging.
  - ✓ But, can cause poor performance due to a large random write operations.
- No Steal policy – don't allow buffer pool frames with uncommited updates to overwrite committed data on DB disk.
  - ✓ Useful for ensuring atomicity without UNDO logging.
  - ✓ But can cause poor performance due to (1)A larger buffer is required; or (2)writing that data to temporary location on non-volatile storage (e.g., swap area).

Buffer Pool Management, Cont.

- Force policy – make sure that every update is on the DB disk before commit.
  - ✓ Provides durability without REDO logging.
  - ✓ But, can cause poor performance due to a large random write operations.
- No Steal policy – don't allow buffer pool frames with uncommited updates to overwrite committed data on DB disk.
  - ✓ Useful for ensuring atomicity without UNDO logging.
  - ✓ But can cause poor performance due to (1)A larger buffer is required; or (2)writing that data to temporary location on non-volatile storage (e.g., swap area).

In practice, even to get Force/No-Steal to work requires some nasty details for handling unexpected failures…

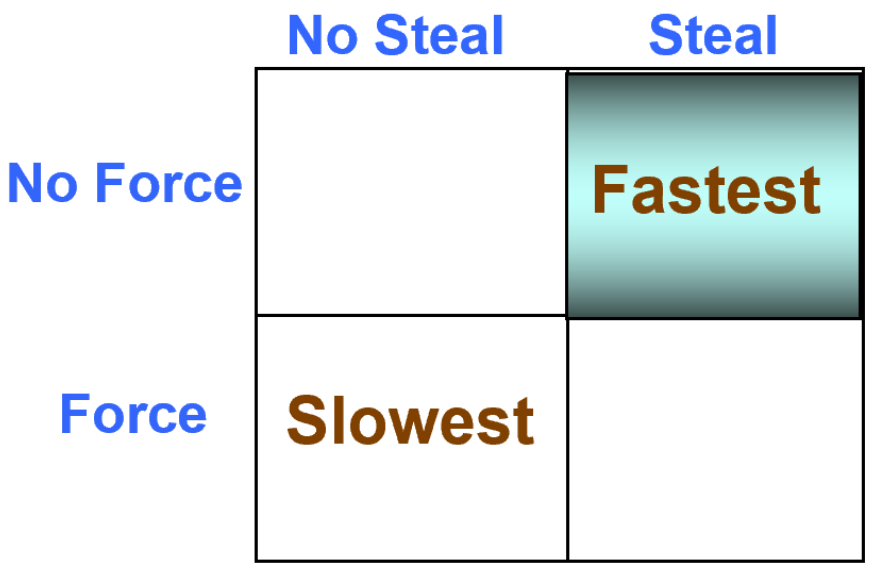
Buffer Pool Replace Policy, Cont.

- No Force
  - What if system crashes before a modified page written by a committed transaction makes it to DB disk?
    - ✓ Write as little as possible, in a convenient place, at commit time, to support REDOing modifications. → WAL Logging.

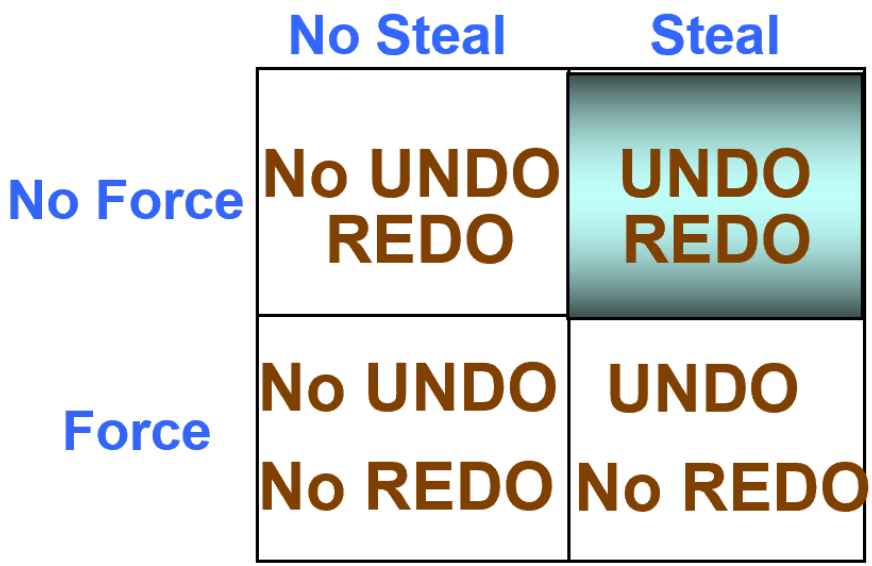
Buffer Pool Replace Policy, Cont.

- No Force
  - What if system crashes before a modified page written by a committed transaction makes it to DB disk?
    - ✓ Write as little as possible, in a convenient place, at commit time, to support REDOing modifications. → WAL Logging.
- Steal
  - What if a transaction that performed updates aborts? → WAL Logging
  - What if system crashes before transaction is finished? → WAL Logging
    - ✓ Must remember the old value of P (to support UNDOing the write to page P).

Buffer Pool Management, Cont.

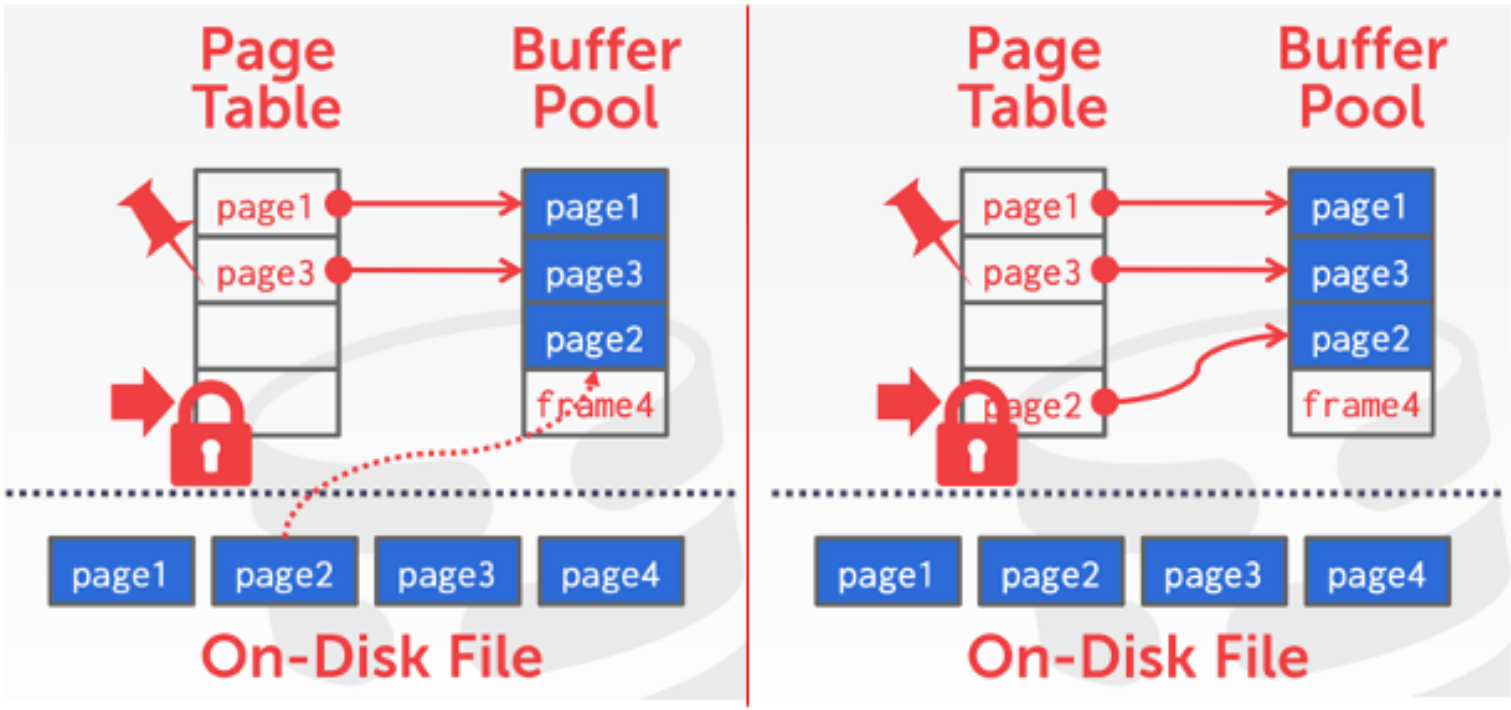


Performance Implications



Log/Recovery Implications

Buffer Pool Management, Cont. [1]



[1] CMU 15-445/645 Fall 2021 Buffer Pool Slide

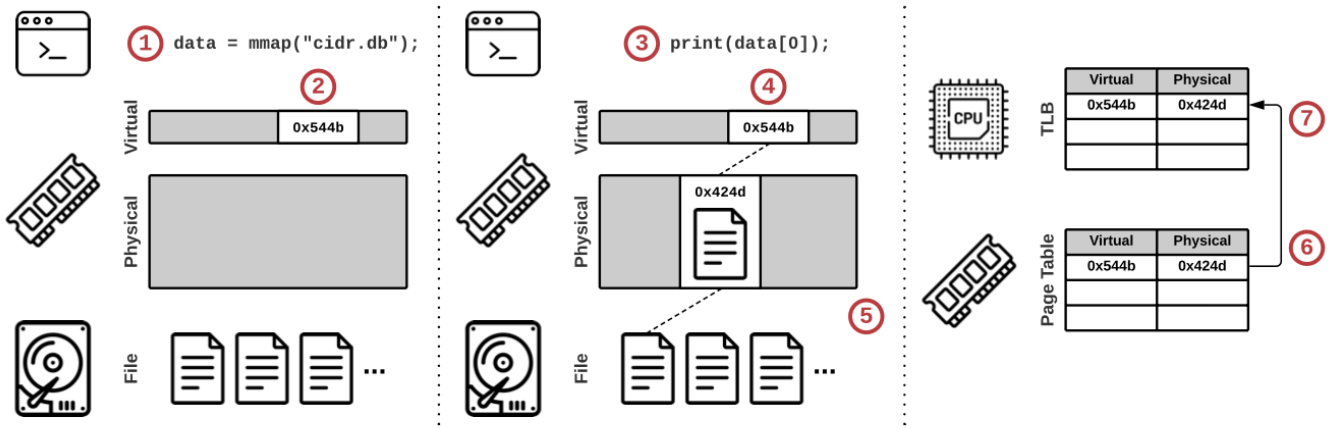
MMAP as Buffer Pool, Cont. [2]

- Memory-mapped (mmap) file I/O is an OS-provided feature.
  - ✓ It maps the contents of a file on secondary storage into a program's virtual address space.

[2] Ethanzjp MMAP Gendata Demo

MMAP as Buffer Pool, Cont. [2]

- Memory-mapped (mmap) file I/O is an OS-provided feature.
  - ✓ It maps the contents of a file on secondary storage into a program's virtual address space.
  - ✓ The program then accesses pages via pointers as if the file resided entirely in memory.

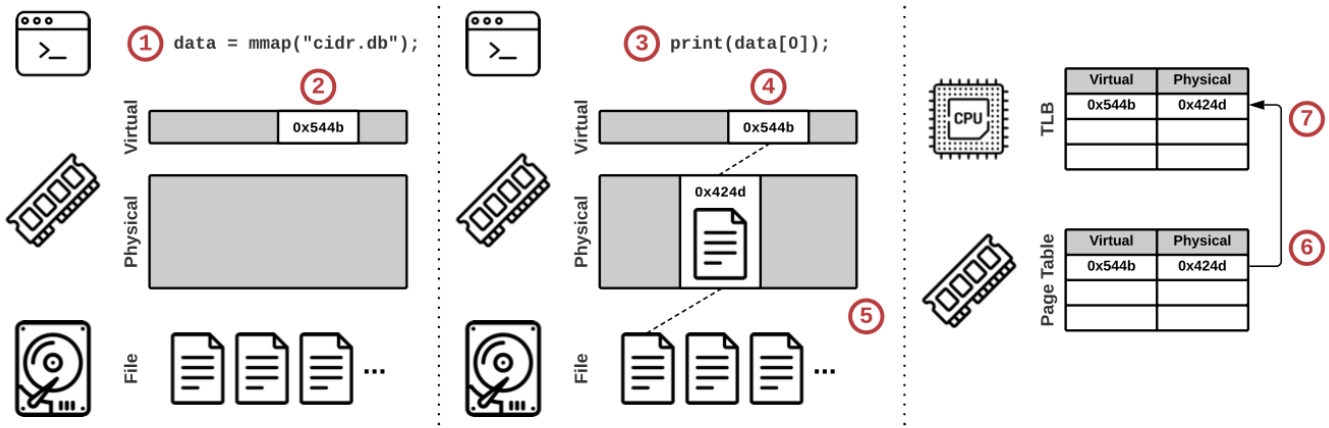


Step-by-step illustration of how a program accesses a file using mmap.

[2] Ethanzjp MMAP Gendata Demo

MMAP as Buffer Pool, Cont. [2]

- Memory-mapped (mmap) file I/O is an OS-provided feature.
  - ✓ It maps the contents of a file on secondary storage into a program's virtual address space.
  - ✓ The program then accesses pages via pointers as if the file resided entirely in memory.
  - ✓ The OS transparently loads pages only when the program references them.

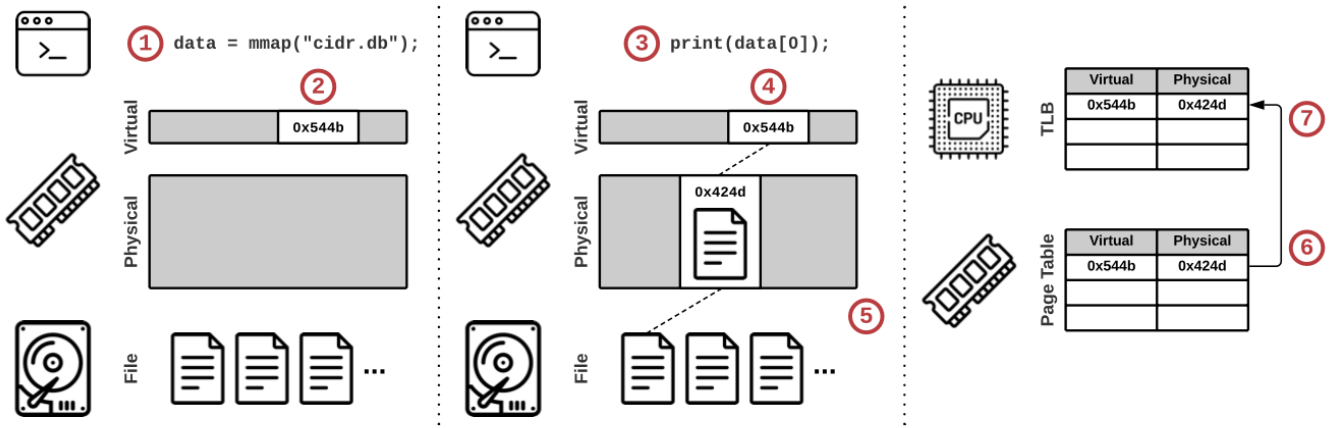


Step-by-step illustration of how a program accesses a file using mmap.

[2] Ethanzjp MMAP Gendata Demo

MMAP as Buffer Pool, Cont. [2]

- Memory-mapped (mmap) file I/O is an OS-provided feature.
  - ✓ It maps the contents of a file on secondary storage into a program's virtual address space.
  - ✓ The program then accesses pages via pointers as if the file resided entirely in memory.
  - ✓ The OS transparently loads pages only when the program references them.
  - ✓ The OS automatically evicts pages if memory fills up.

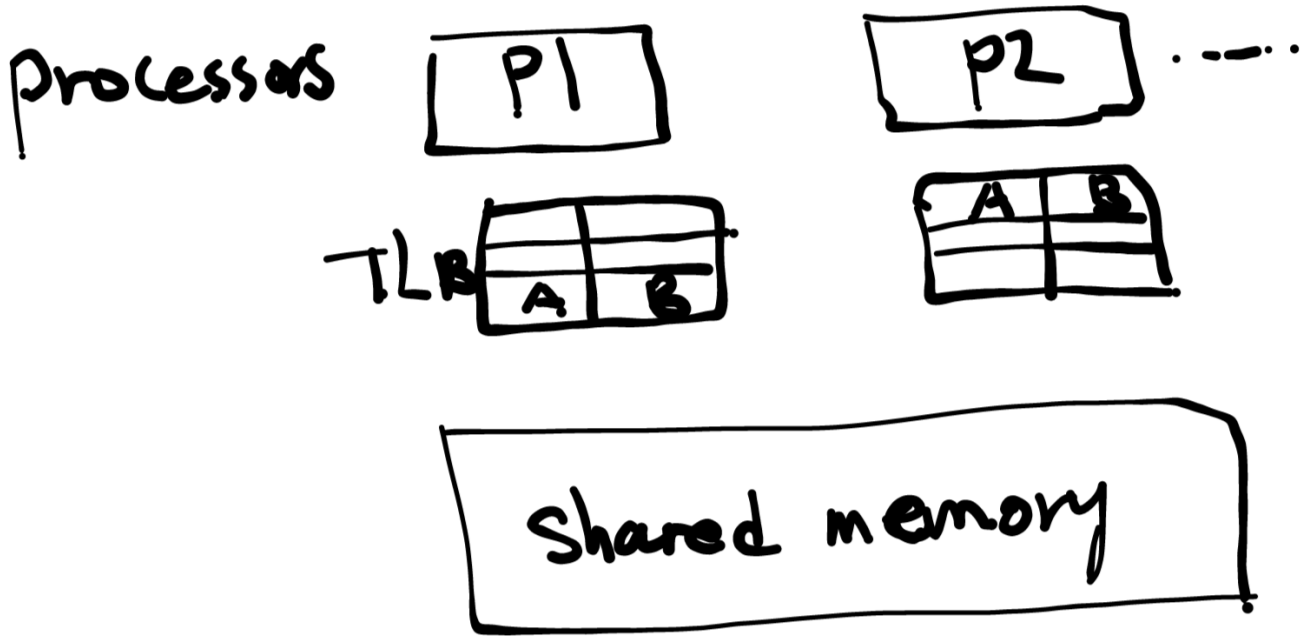


Step-by-step illustration of how a program accesses a file using mmap.

[2] Ethanzjp MMAP Gendata Demo

TLB shootdown

- Shared Memory Model[3]



[3] Ethanzjp Shared Memory Demo

Comparsion of Buffer Pool and MMAP

- Buffer Pool
  - ✓ The DBMS maintaining complete control over how and when it transfers pages.
- MMAP
  - ✓ The OS handles all necessary paging behind the scenes rather than the DBMS's buffer pool.
- Stonebraker 1981 opinion [4]

> However, many DBMSs including INGRES [20] and System R [4] choose to put a DBMS managed buffer pool in user space to reduce overhead. Hence, each of these systems has gone to the trouble of constructing its own buffer pool manager to enhance performance.

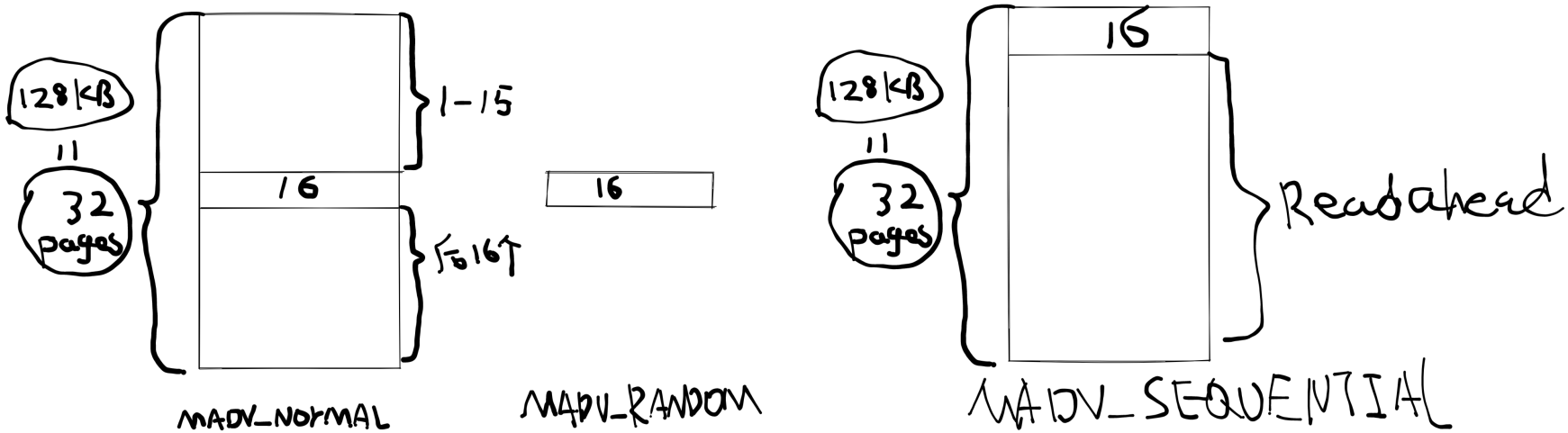
[4] 1981 Stonebraker's Paper

What's the problems?

- Reading the data from disk
- Concurrency between different threads reading the same data
- Caching and buffer management
- Eviction of pages from memory
- Playing nice with other processes in the machine
- Tracking dirty pages and writing to disk

POSIX API, Cont.

- mmap [5]
- madvise hints to the OS about expected data access patterns[6]



- mlock[7] allows DBMS pin memory. But OS is permitted to flush dirty pages to the backing file at any time, even if the page is pinned.
- msync[8] explicitly flushes the specified memory range to secondary storage.

[5] mmap man7 page
[6] madvise man7 page
[7] mlock man7 page
[8] msync man7 page

Modern MMAP-based DBMS

DBMS	MMAP Use	Details
MonetDB	2002–	[12, 21]
MongoDB	2009–2019	[14, 3]
LevelDB	2011–	[5]
LMDB	2011–	[20]
SQLite	2013–	[7]
SingleStore	2013–2015	[32]
QuestDB	2014–	[34]
RavenDB	2014–	[4]
InfluxDB	2015–2020	[8, 1]
WiredTiger	2020–	[17]

[9] LevelDB Snapshot Demo

Modern MMAP-based DBMS

DBMS	MMAP Use	Details
MonetDB	2002–	[12, 21]
MongoDB	2009–2019	[14, 3]
LevelDB	2011–	[5]
LMDB	2011–	[20]
SQLite	2013–	[7]
SingleStore	2013–2015	[32]
QuestDB	2014–	[34]
RavenDB	2014–	[4]
InfluxDB	2015–2020	[8, 1]
WiredTiger	2020–	[17]

- MonetDB use WiredTiger and MMAPv1 (optional) as storage engine.
- InfluxDB replaced mmap after observing I/O spikes for writes when a database grew larger than a few GB in size.

[9] LevelDB Snapshot Demo

Modern MMAP-based DBMS

DBMS	MMAP Use	Details
MonetDB	2002–	[12, 21]
MongoDB	2009–2019	[14, 3]
LevelDB	2011–	[5]
LMDB	2011–	[20]
SQLite	2013–	[7]
SingleStore	2013–2015	[32]
QuestDB	2014–	[34]
RavenDB	2014–	[4]
InfluxDB	2015–2020	[8, 1]
WiredTiger	2020–	[17]

- MonetDB use WiredTiger and MMAPv1 (optional) as storage engine.
- InfluxDB replaced mmap after observing I/O spikes for writes when a database grew larger than a few GB in size.
- SingleStore removed mmap-based file I/O after encountering poor performance on simple sequential scan queries.

[9] LevelDB Snapshot Demo

Modern MMAP-based DBMS

DBMS	MMAP Use	Details
MonetDB	2002–	[12, 21]
MongoDB	2009–2019	[14, 3]
LevelDB	2011–	[5]
LMDB	2011–	[20]
SQLite	2013–	[7]
SingleStore	2013–2015	[32]
QuestDB	2014–	[34]
RavenDB	2014–	[4]
InfluxDB	2015–2020	[8, 1]
WiredTiger	2020–	[17]

- MonetDB use WiredTiger and MMAPv1 (optional) as storage engine.
- InfluxDB replaced mmap after observing I/O spikes for writes when a database grew larger than a few GB in size.
- SingleStore removed mmap-based file I/O after encountering poor performance on simple sequential scan queries.
- RocksDB replace mmap as a fork of LevelDB [9].

[9] LevelDB Snapshot Demo

What is the truth?

The DBMS seems no longer needs to manage its own buffer pool, as it cedes this responsibility to the OS.

Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW

Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW
    - ✓ MAP_PRIVATE to enable OS CoW.

Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW
    - ✓ MAP_PRIVATE to enable OS CoW.
    - ✓ The DBMS modifies the affected pages in the private workspace.

Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW
    - ✓ MAP_PRIVATE to enable OS CoW.
    - ✓ The DBMS modifies the affected pages in the private workspace.
    - ✓ To provide durability, the DBMS must use a write-ahead log (WAL) to record changes.

Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW
    - ✓ MAP_PRIVATE to enable OS CoW.
    - ✓ The DBMS modifies the affected pages in the private workspace.
    - ✓ To provide durability, the DBMS must use a write-ahead log (WAL) to record changes.
    - ✓ DBMS applies the committed changes to the primary copy using msync.

Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW
    - ✓ MAP_PRIVATE to enable OS CoW.
    - ✓ The DBMS modifies the affected pages in the private workspace.
    - ✓ To provide durability, the DBMS must use a write-ahead log (WAL) to record changes.
    - ✓ DBMS applies the committed changes to the primary copy using msync.
  - User space CoW

Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW
    - ✓ MAP_PRIVATE to enable OS CoW.
    - ✓ The DBMS modifies the affected pages in the private workspace.
    - ✓ To provide durability, the DBMS must use a write-ahead log (WAL) to record changes.
    - ✓ DBMS applies the committed changes to the primary copy using msync.
  - User space CoW
    - ✓ Copy the mmap-backed memory page to user buffer.

Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW
    - ✓ MAP_PRIVATE to enable OS CoW.
    - ✓ The DBMS modifies the affected pages in the private workspace.
    - ✓ To provide durability, the DBMS must use a write-ahead log (WAL) to record changes.
    - ✓ DBMS applies the committed changes to the primary copy using msync.
  - User space CoW
    - ✓ Copy the mmap-backed memory page to user buffer.
    - ✓ Update and recording the WAL logging.

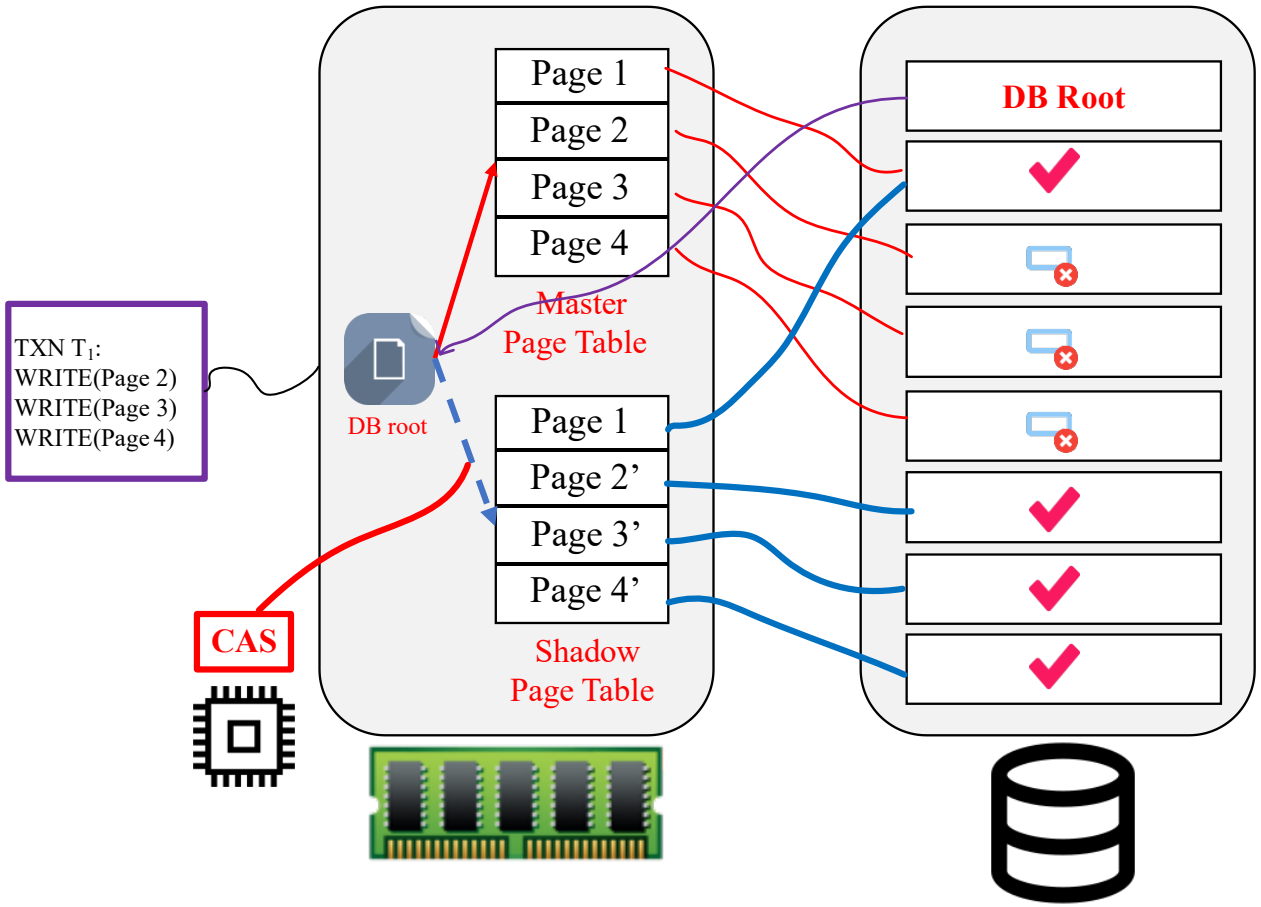
Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW
    - ✓ MAP_PRIVATE to enable OS CoW.
    - ✓ The DBMS modifies the affected pages in the private workspace.
    - ✓ To provide durability, the DBMS must use a write-ahead log (WAL) to record changes.
    - ✓ DBMS applies the committed changes to the primary copy using msync.
  - User space CoW
    - ✓ Copy the mmap-backed memory page to user buffer.
    - ✓ Update and recording the WAL logging.
    - ✓ Copy the modified pages back to the mmap-backed memory.

Transactional Safety

- The challenges inherent with guaranteeing transactional safety of modified pages in mmap-based DBMSs are well-known.
  - Due to transparent paging, the OS can flush a dirty page to secondary storage at any time, irrespective of whether the writing transaction has committed.
  - The DBMS cannot prevent these flushes and receives no warning when they occur.
- Three categories for handling updates:
  - OS CoW
    - ✓ MAP_PRIVATE to enable OS CoW.
    - ✓ The DBMS modifies the affected pages in the private workspace.
    - ✓ To provide durability, the DBMS must use a write-ahead log (WAL) to record changes.
    - ✓ DBMS applies the committed changes to the primary copy using msync.
  - User space CoW
    - ✓ Copy the mmap-backed memory page to user buffer.
    - ✓ Update and recording the WAL logging.
    - ✓ Copy the modified pages back to the mmap-backed memory.
  - Shadow Paging

System R's Shadow paging



- Master: Contains only changes from committed txns.
- Shadow: Temporary db with changes made from uncommitted txns.

IO Stalls

- Accessing any page could result in an unexpected I/O stall because the DBMS cannot know whether the page is in memory.
  - ✓ Pinning memory.
  - ✓ mlock the memory.
  - ✓ madvise, but os is free ignore the advise.

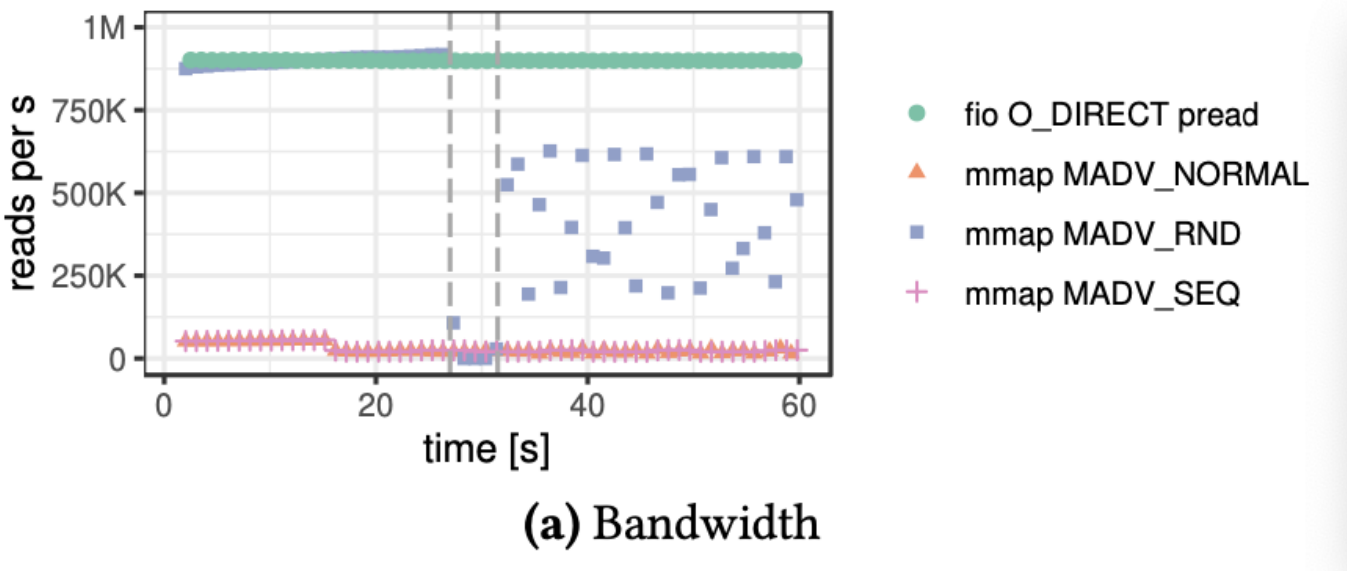
Error Handling

- page-level checksums
- gracefully handling I/O errors

Performance Issues

- page table contention
- single-threaded page evictionand for larger-than- memory DBMS workloads on high-bandwidth secondary storage devices.
- TLB shootdowns.

Random Reads on Bandwidth[10]

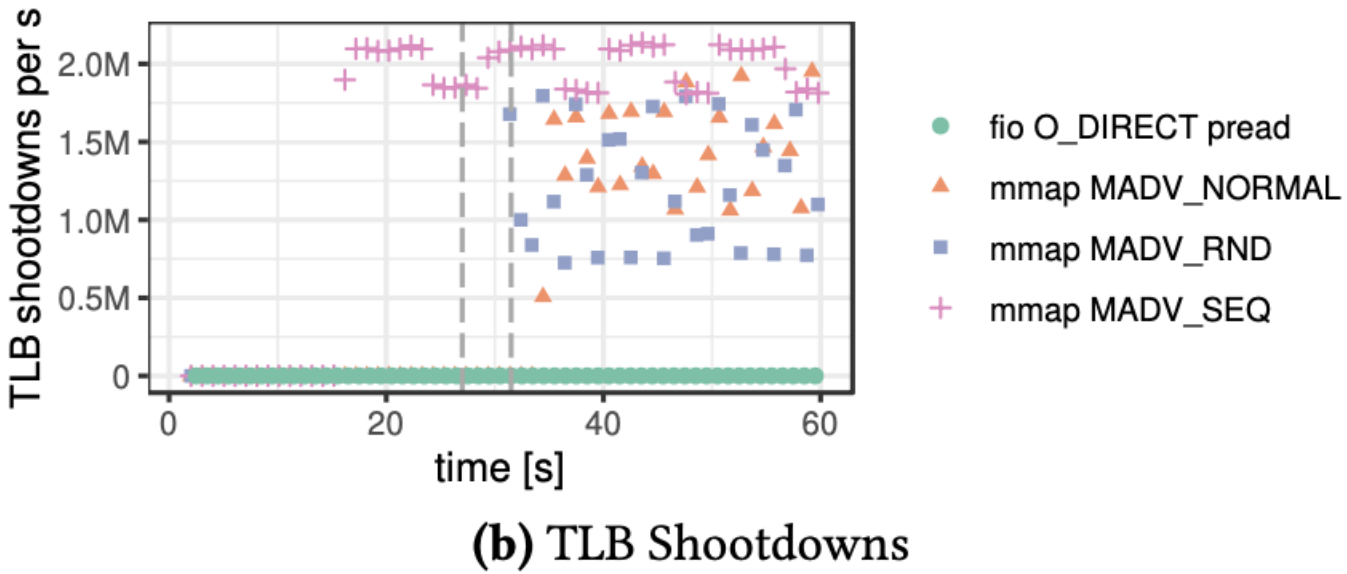


(a) Bandwidth

- Random access pattern over a 2 TB SSD range to simulate a larger-than-memory OLTP workload.
- The page cache had only 100 GB of memory, 95% of all accesses resulted in page faults
- fio baseline exhibited stable performance and achieved close to 900K reads per second

[10] mmapbenchmark

Random Reads on TLB Shootdown



(b) TLB Shootdowns

- we measured using /proc/interrupt

Sequential Scan on Bandwidth

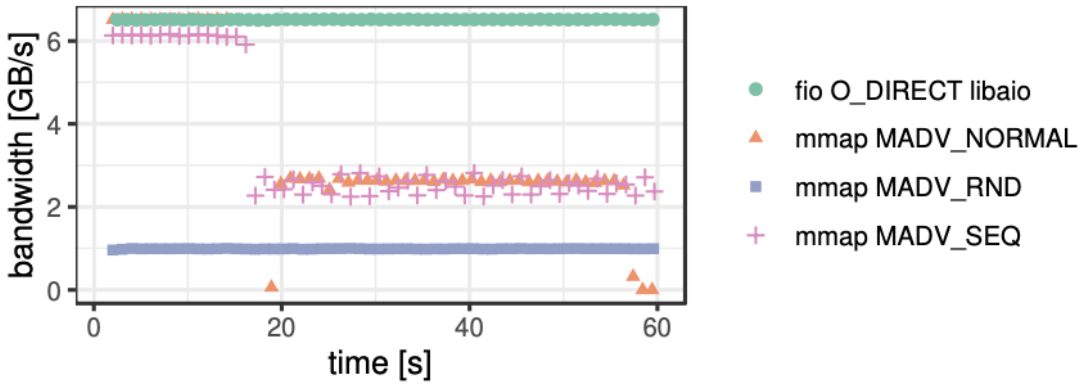


Figure 3: Sequential Scan – 1 SSD (mmap: 20 threads; fio: libaio, 1 thread, iodepth 256)

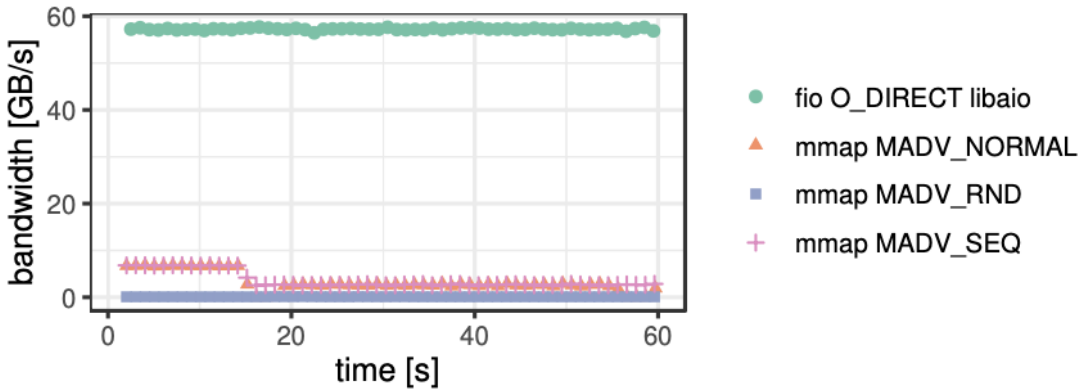


Figure 4: Sequential Scan – 10 SSDs (mmap: 20 threads; fio: libaio, 4 threads, iodepth 256)

Conclusions

- mmap is not a suitable replacement for a traditional buffer pool.
- When you should not use mmap in your DBMS:
  - You need to perform updates in a transactionally safe fashion.

Conclusions

- mmap is not a suitable replacement for a traditional buffer pool.
- When you should not use mmap in your DBMS:
  - You need to perform updates in a transactionally safe fashion.
  - You want to handle page faults without blocking on slow I/O or need explicit control over what data is in memory.

Conclusions

- mmap is not a suitable replacement for a traditional buffer pool.
- When you should not use mmap in your DBMS:
  - You need to perform updates in a transactionally safe fashion.
  - You want to handle page faults without blocking on slow I/O or need explicit control over what data is in memory.
  - You care about error handling and need to return correc results.

Conclusions

- mmap is not a suitable replacement for a traditional buffer pool.
- When you should not use mmap in your DBMS:
  - You need to perform updates in a transactionally safe fashion.
  - You want to handle page faults without blocking on slow I/O or need explicit control over what data is in memory.
  - You care about error handling and need to return correc results.
  - You require high throughput on fast persistent storage devices.

Conclusions

- mmap is not a suitable replacement for a traditional buffer pool.
- When you should not use mmap in your DBMS:
  - You need to perform updates in a transactionally safe fashion.
  - You want to handle page faults without blocking on slow I/O or need explicit control over what data is in memory.
  - You care about error handling and need to return correc results.
  - You require high throughput on fast persistent storage devices.
- When you should maybe use mmap in your DBMS:

Conclusions

- mmap is not a suitable replacement for a traditional buffer pool.
- When you should not use mmap in your DBMS:
  - ∗ You need to perform updates in a transactionally safe fashion.
  - ∗ You want to handle page faults without blocking on slow I/O or need explicit control over what data is in memory.
  - ∗ You care about error handling and need to return correc results.
  - ∗ You require high throughput on fast persistent storage devices.
- When you should maybe use mmap in your DBMS:
  - ✓ Your working set (or the entire database) fits in memory and the workload is read-only.

Conclusions

- mmap is not a suitable replacement for a traditional buffer pool.
- When you should not use mmap in your DBMS:
  - You need to perform updates in a transactionally safe fashion.
  - You want to handle page faults without blocking on slow I/O or need explicit control over what data is in memory.
  - You care about error handling and need to return correc results.
  - You require high throughput on fast persistent storage devices.
- When you should maybe use mmap in your DBMS:
  - ✓ Your working set (or the entire database) fits in memory and the workload is read-only.
  - ✓ You need to rush a product to the market and do not care about data consistency or long-term engineering headaches.

Conclusions

- mmap is not a suitable replacement for a traditional buffer pool.
- When you should not use mmap in your DBMS:
  - ∗ You need to perform updates in a transactionally safe fashion.
  - ∗ You want to handle page faults without blocking on slow I/O or need explicit control over what data is in memory.
  - ∗ You care about error handling and need to return correc results.
  - ∗ You require high throughput on fast persistent storage devices.
- When you should maybe use mmap in your DBMS:
  - ✓ Your working set (or the entire database) fits in memory and the workload is read-only.
  - ✓ You need to rush a product to the market and do not care about data consistency or long-term engineering headaches.
  - ✓ Otherwise, never.

References

 RavenDB Response

 Community Comments

Thank you!
Welcome for any questions!

Junpeng Zhu
Greenplum, VMware, Inc.