University of Cyprus

Are you sure you want to use MMAP in your database management system

Presenter: Antonis Louca

Arguments

- Because of MMAP benefits
  - DBMS developers are seduced to use it as a buffer pool alternative
- Severe **correctness and performance issues** are **unapparent**
- MMAP is not suitable to replace traditional buffer pool

Potential benefits to use MMAP

- Easy to use
- Low engineering cost
- Use pointers to access data
  - OS handles space management transparently
  - Returns pointers to OS's page cache
- Lower performance overhead
  - No cost from read/write system calls
- Lower total memory consumption

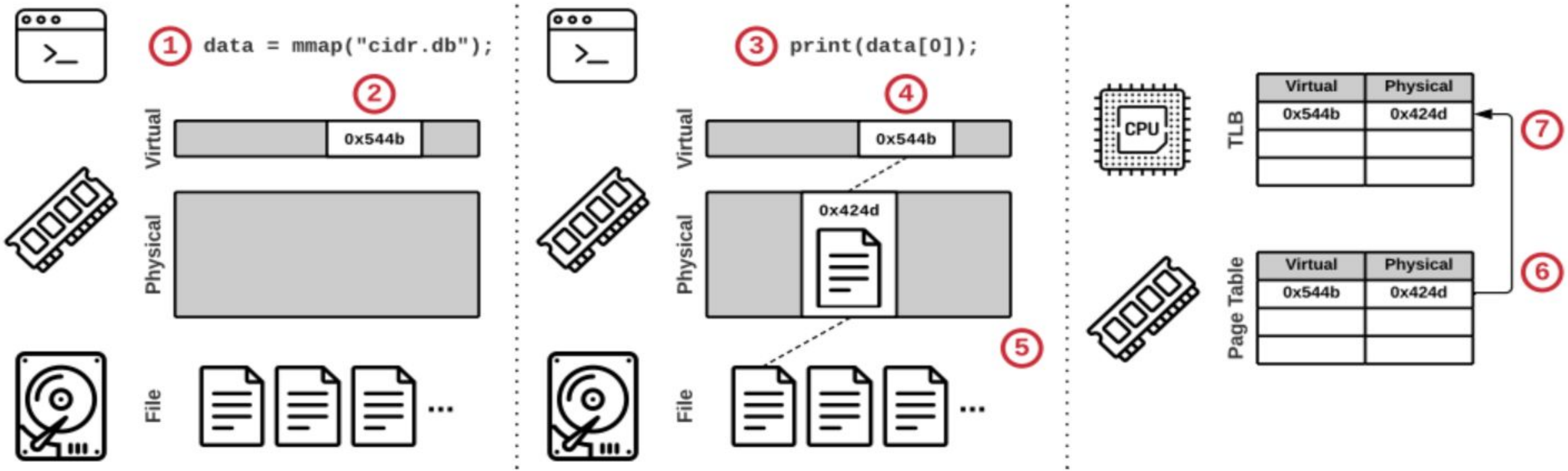
Buffer Pool vs MMAP

- Use of a buffer pool
  - Component which interacts with secondary storage
  - Moves pages between secondary storage and main memory
  - Provides illusion that whole database exists in main memory
  - Provides complete control of page fetching and eviction to DBMS

Buffer Pool vs MMAP

- Use of MMAP
    - MMAP is a feature provided by the OS
    - DBMS gives responsibility of data movement to OS
    - OS keeps its own file mapping and page cache
    - MMAP maps a file from secondary storage to DBMS's virtual address space
    - OS loads pages **lazily** when DBMS accesses them
        - **Pages loaded only when referenced**
            - **Page loading** and **eviction** is done **transparently** by the OS

MMAP overview



```
data = mmap("cidr.db");
```

```
print(data[0]);
```

TLB

Virtual	Physical
0x544b	0x424d

Page Table

Virtual	Physical
0x544b	0x424d

Figure 1: Step-by-step illustration of how a program accesses a file using `mmap`.

Posix API system calls

- **mmap**
  - Causes the OS to map file into DBMS’s virtual address space
- **madvice**
  - Gives the ability to give hints to the OS about data access patterns.
  - File granularity or page range granularity
- **mlock**
  - Allows DBMS to pin pages in memory, so that OS never evicts them
- **msync**
  - Flushes memory range to secondary storage

Some Databases that used MMAP

- ❑ MongoDB
- ❑ InfluxDB
- ❑ SingleStore
- ❑ TitleDB
- ❑ Scylla

Problems with MMAP

Problem 1: Transactional Safety

- **DBMS must ensure that transparent paging does not violate transactional safety guarantees**
  - **OS due to transparent paging can flush a dirty page in secondary storage at any time, without knowing if transaction has committed or not.**
- Update handling methods:
  - OS copy-on-write
  - User space copy-on-write
  - Shadow paging

Problem 1: Transactional Safety

- **OS copy-on-write**
  - Create two copies of the DB file with mmap, initially in the same physical pages
  - 1 copy is a primary copy and the other is a private workspace
  - Update → modify affected pages in private work space
    - Transparently copy to new physical pages
    - Remap virtual memory addresses to the copies and apply changes
    - Primary copy does not see these changes and won't be written
  - Need to use Write-ahead-log (WAL) to record changes
  - When transaction commits the DBMS flushes the WAL records on secondary storage
  - Secondary thread applies changes to primary copy

Problem 1: Transactional Safety

- **OS copy-on-write**
  - Problems:
    - Must ensure updates for committed transactions propagated to primary copy before running conflicting transactions
    - Private workspace grows as number of updates grow and result in two full copies in memory

Problem 1: Transactional Safety

- **User space copy-on-write**
  - Manually copy affected pages from mmap-backed memory to a buffer in user space
  - Update → only the copies and create WAL records
  - When WAL records are written in secondary storage →updates can propagate to mmap-backed pages.
- **Shadow paging**
  - Maintains two copies shadow and primary
  - Both are backed by mmap
  - On update
    - DBMS copies affected pages to shadow copy, and applies changes
    - Flush modified pages to secondary storage
    - Shadow copy is the new primary and primary the new shadow copy

Problem 2: I/O stalls

- **MMAP does not support asynchronous reads**
    - Traditional buffer pool can use asynchronous read requests for non-contiguous pages
    - Avoids thread blocking
    - Masks latency
- Read-only queries can trigger blocking pages faults
    - OS transparently evicts pages
    - When read-only queries access evicted pages can cause I/O stalls
- Any page access can cause I/O stalls

Problem 2: I/O stalls - Mitigation

- **mlock to pin pages** accessed in the future
  - OS restricts the memory amount a process can lock
  - Can cause problems to other running processes or the OS
- **madvice to hint the OS** about expected access patterns
  - Less involved than mlock with less control
  - Providing the wrong hint can lead to serious performance overheads
- Spawn **other threads to handle page prefetching**
  - These threads will block in case of page fault event
  - Main thread does not block
  - Additional complexity

Problem 3: Error Handling

- **Data integrity mandates error handling**
- Page level checksums help in data corruption detection
  - Read page from disk → validate contents using the stored checksum
- **With mmap DBMS needs to validates page on every access**
  - OS may have evicted the page at some point
- DBMS are written in memory unsafe languages →pointer errors cause corruption
- Error handling becomes more difficult

Problem 4: Performance Issues

- mmap has serious **performance bottlenecks**
    - **can only be overcomed with OS redesign**

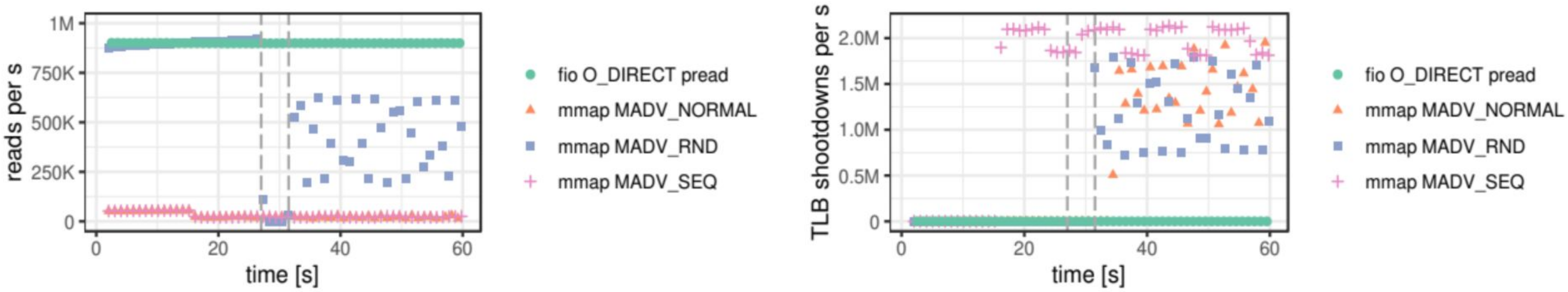
Problem 4: Performance Issues

- Three main performance issues
  - **Page table contention**
  - **Single threaded page eviction**
  - **TLB shootdowns**
- The first two problems are mitigated with relative ease
- TLB shootdowns are tricker
  - Local TLB flushing is inexpensive
  - **Synchronization of remote TLBs requires thousands of cycles**
    - Microarchitectural changes
    - Extensive OS modifications

Experimental Analysis

- Evaluate performance of traditional techniques against MMAP
- Used the **fio** storage **benchmarking tool**
- MMAP with **random**, **normal**, **sequential** hints
- Evaluated two **access patterns**
  - **Random reads**
  - **Sequential scan**

Experimental Analysis - Random Reads



(a) Bandwidth

(b) TLB Shootdowns

Figure 2: Random Reads – 1 SSD (100 threads)

Experimental Analysis - Sequential Scan

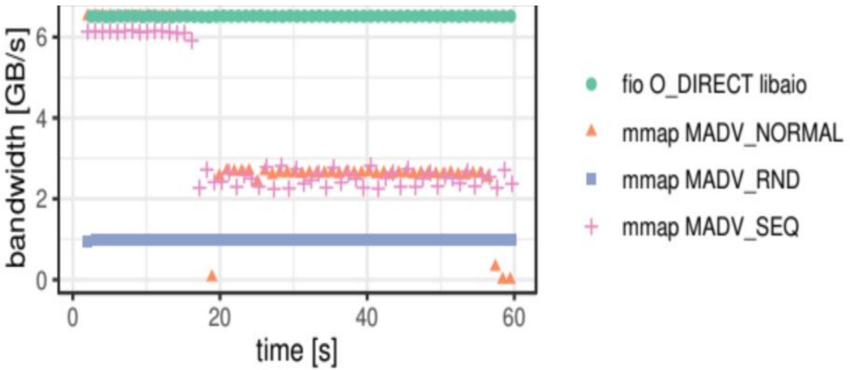


Figure 3: Sequential Scan – 1 SSD (mmap: 20 threads; `fio`: `libaio`, 1 thread, `iodepth` 256)

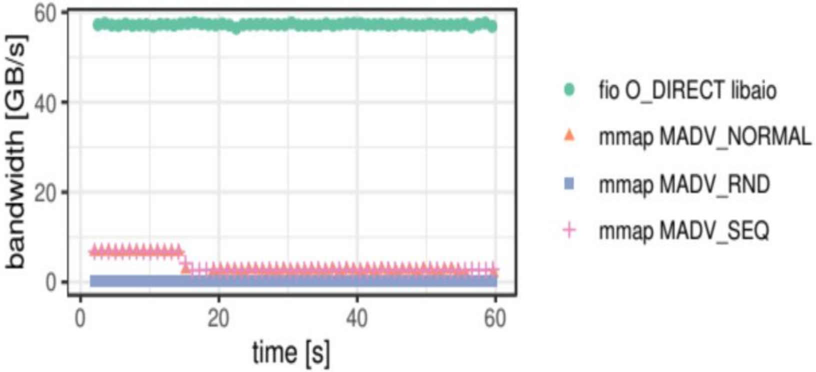


Figure 4: Sequential Scan – 10 SSDs (mmap: 20 threads; `fio`: `libaio`, 4 threads, `iodepth` 256)

Conclusions

- Should use mmap only if entire DB fits in memory
- Should not use mmap if
  - Updates on transactionally safe fashion is required
  - Page fault handling without blocking
  - Error handling
  - Need high throughput on fast persistent storage devices

References

- Announcing InfluxDB IOx - The Future Core of InfluxDB Built with Rust and Arrow. https://www.influxdata.com/blog/announcing-influxdb-iox/.
- fio: Flexible I/O Tester. https://github.com/axboe/fio.
- MongoDB MMAPv1 Storage Engine. https://docs.mongodb.com/v4.0/core/ mmapv1/.
- The InfluxDB storage engine and the Time-Structured Merge Tree. https://docs. influxdata.com/influxdb/v1.8/concepts/storage_engine/.
- C. Villavieja, V. Karakostas, L. Vilanova, Y. Etsion, A. Ramírez, A. Mendelson, N. Navarro, A. Cristal, and O. S. Unsal. DiDi: Mitigating the Performance Impact of TLB Shootdowns Using a Shared TLB Directory. In PACT, pages 340–349, 2011
- https://en.wikipedia.org/wiki/Write-ahead_logging
- https://man7.org/linux/man-pages/man2/mmap.2.html
- https://en.wikipedia.org/wiki/Translation_lookaside_buffer

Thank you for your attention

Any Questions?