DATA ANALYSIS AND DEEP LEARNING

CS 8803 // FALL 2018 //
Sneha Venkatachalam

Main Memory Database Systems: An Overview

IEEE 1992

CREATING THE NEXT®

TODAY'S PAPER

"Main Memory Database Systems: An Overview "

AUTHORS

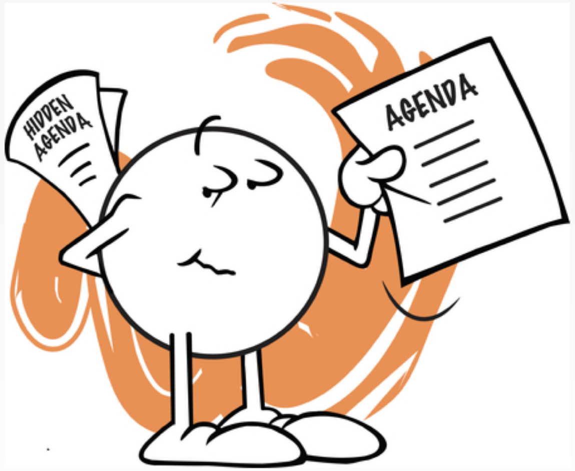
Hector Garcia-Molina and Kenneth Salem

AREAS OF FOCUS

Access methods, application programming interface, commit processing, concurrency control, data clustering, data representation, main memory database system (MMDB), query processing, recovery.

TODAY'S AGENDA

- Concepts
- Problem Overview
- Key Idea
- Technical Details
- Evaluation
- Discussion



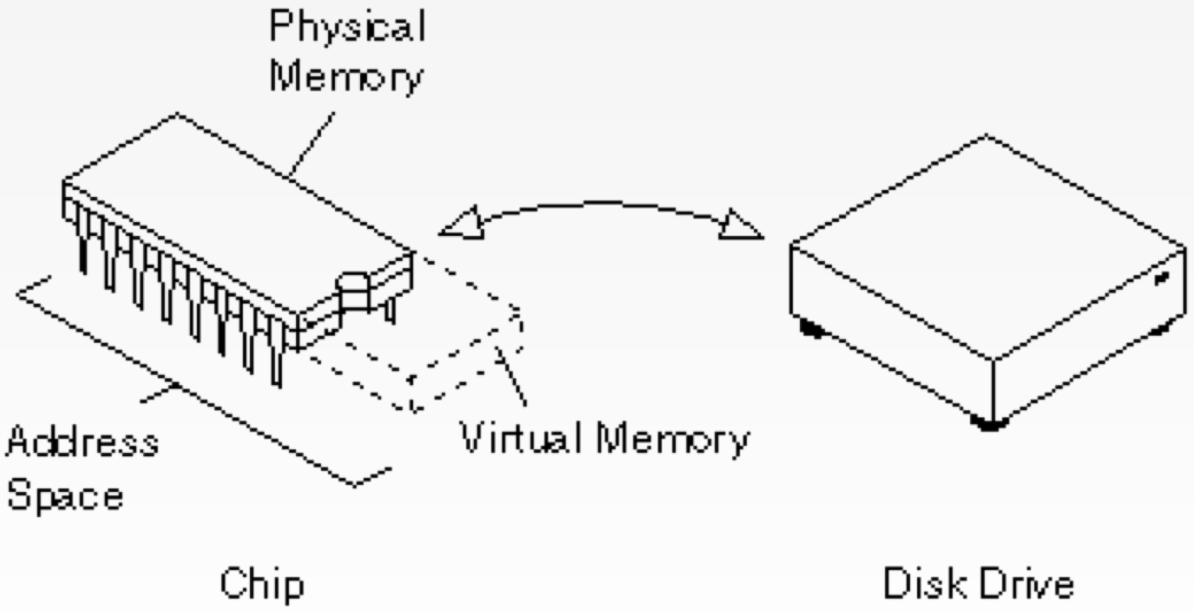
OVERVIEW

- Memory resident database systems (MMDB's) store their data in main physical memory and provide very high-speed access
- Conventional database systems (DRDB) are optimized for the particular characteristics of disk storage mechanisms.
- Memory resident systems, on the other hand, use different optimizations to structure and organize data, as well as to make it reliable
- This paper surveys the major memory residence optimizations and briefly discusses some of the memory resident systems that have been designed or implemented

DIFFERENCE BETWEEN MEMORY AND DISK

- The access time for main memory is orders of magnitude less than for disk storage.
- Main memory is normally volatile, while disk storage is not.
- Disks have a high, fixed cost per access as they are block-oriented storage device, however main memory is not block oriented
- The layout of data on a disk is much more critical than the layout of data in main memory, since sequential access to a disk is faster than random access
- Main memory is normally directly accessible by the processor(s), while disks are not, which makes data in memory more vulnerable to software errors

MEMORY AND DISK



Is it reasonable to assume that the entire database fits in main memory?

- Yes, for some applications:
  1. Cases where database is of limited size or is growing at a slower rate than memory capacities are growing
     - Ex. Database containing employee data.
       - It is reasonable to expect that memory can hold a few hundred or thousand bytes per employee or customer
  2. Real-time applications where data must be memory resident to meet the real-time constraints
     - Ex1. Telecommunications: 800 telephone numbers need to be translated to real numbers
     - Ex2. Radar tracking: Signatures of objects need to be matched against a database of known aircraft

Is it reasonable to assume that the entire database fits in main memory?

- No for cases where the database does not fit in memory
  - Ex. An application with satellite image data
  - DRDB will continue to be important here
- However, these applications can be classified into ’hot’ (accessed frequently) and ‘cold’ (accessed rarely) data
  - Data can be partitioned into one or more logical databases, and the hottest one can be stored in main memory
  - A collection of databases is now managed by both MMDB and DRDB
  - Ex. In banking, account records (ex., containing balances) are hot; customer records (ex., containing address, mother’s maiden name) are colder
  - IMS database system: Provides Fast Path for memory resident data, and conventional IMS for the rest

What is the difference between a MMDB and a DRDB with a very large cache?

- Large DRDB cache enables storing copies of datasets in memory at all times
- This does not take full advantage of the memory
- Ex. Say an application wishes to access a given tuple
  - The disk address will have to be computed
  - The buffer manager will be invoked to check if the corresponding block is in memory
  - Once the block is found, the tuple will be copied into an application tuple buffer, where it is actually examined.
  - Clearly, if the record will always be in memory, it is more efficient to refer to it by its memory address

What is the difference between a MMDB and a DRDB with a very large cache?

- Some DRDB and some object-oriented storage systems (OOSS) are beginning to recognize that with large caches some of their data will reside often in memory, and are beginning to implement some of the inmemory optimizations of MMDB
  - Ex. Some new systems convert a tuple or object into an in-memory representation and give applications a direct pointer to it
  - This is called “swizzling”
- In future, the differences between a MMDB and DRDB might disappear
- Any good database management system will recognize and exploit the fact that some data will reside permanently in memory and should be managed accordingly

Can we assume that main memory is nonvolatile and reliable by introducing special purpose hardware?

- Performance improvement; No crash recovery code
- There is no “yes” or “no” answer
- Memory can be made more reliable by techniques
  - Battery-backed up memory boards
  - Uninterruptable power supplies
  - Error detecting and correcting memory
  - Triple modular redundancy
- However, this only reduces the probability of media failure
- Thus one will always have to have a backup copy of the database, probably on disk

FACTORS AFFECTING FREQUENCY OF BACKUPS FOR MMDB

- Memory is directly accessible by the processor and is more vulnerable to operating system errors.
  - Hence, system crashes will lead to loss of memory
- When a memory board fails, typically the entire machine must be powered down, losing the entire database
  - A recent backup is required as recovery of the data will be much more time consuming otherwise
- Battery backed memory, or uninterruptable power supplies (UPS) are “active” devices and lead to higher probability of data loss than do disks
  - A UPS can run out of gas or can overheat.
  - Batteries can leak or lose their charge.

VIDEO

https://www.youtube.com/watch?v=p3q5zWCw8J4

IMPACT OF MEMORY RESIDENT DATA Concurrency Control

- Access to main memory is so much faster than disk access
- Hence, we can expect transactions to complete more quickly in a main memory system
- In systems that use lock-based concurrency controls, this means that locks will not be held as long
- Therefore, lock contention may not be as important as it is when the data is disk resident.

IMPACT OF MEMORY RESIDENT DATA
Concurrency Control

- The actual implementation of the locking mechanism can also be optimized for memory residence of the objects to be locked
- In a conventional system, locks are implemented via a hash table that contains entries for the objects currently locked
- The objects themselves (on disk) contain no lock information
- If the objects are in memory, we may be able to afford a small number of bits in them to represent their lock status

IMPACT OF MEMORY RESIDENT DATA
Commit Processing

- To protect against media failures, it is necessary to have a backup copy and keep a log of transaction activity
- The need for a stable log threatens to undermine the performance advantages that can be achieved with memory resident data
- Logging can impact response time, since each transaction must wait for at least one stable write before committing
- Logging can also affect throughput if the log becomes a bottleneck
- Several solutions have been suggested for this problem

IMPACT OF MEMORY RESIDENT DATA
Commit Processing

- A small amount of stable main memory can be used to hold a portion of the log
- A transaction is committed by writing its log information into the stable memory (relatively fast)
- A special process or processor is then responsible for copying data from the stable memory to the log disks
- This can eliminate the response time problem, since transactions need never wait for disk operations

IMPACT OF MEMORY RESIDENT DATA

Commit Processing

- In case stable memory is not available for the log tail, transactions can be pre-committed
- Pre-committing is accomplished by releasing a transaction's locks as soon as its log record is placed in the log, without waiting for the information to be propagated to the disk
- The sequential nature of the log ensures that transactions cannot commit before others on which they depend.
- This may reduce the blocking delays (and hence, the response time) of other, concurrent transactions

IMPACT OF MEMORY RESIDENT DATA
Commit Processing

- A technique called group commits can be used to relieve a log bottleneck
- Under group commit, a transaction’s log record need not be sent to the log disk as soon as it commits
- Instead, the records of several transactions are allowed to accumulate in memory
- When enough have accumulated (ex., when a page is full), all are flushed to the log disk in a single disk operation
- Group commit reduces the total number of operations performed by the log disks since a single operation commits multiple transactions

IMPACT OF MEMORY RESIDENT DATA

Access Methods

- A wide variety of index structures have been proposed and evaluated for main memory databases
- These include various forms of hashing and of trees
- Hashing provides fast lookup and update, but not as space-efficient as a tree, and does not support range queries well.
- Trees such as the T-Tree have been designed explicitly for memory-resident databases
- Index structures can store pointers to the indexed data, rather than the data itself
- This eliminates the problem of storing variable length fields in an index and saves space as long as the pointers are smaller than the data they point to

VIDEO

https://www.youtube.com/watch?v=TQQ2gYftnqY

IMPACT OF MEMORY RESIDENT DATA
Data Representation

- Main memory databases take advantage of efficient pointer following for data representation
- Relational tuples can be represented as a set of pointers to data values
- The use of pointers is space efficient when large values appear multiple times in the database, since the actual value needs to only be stored once
- Pointers also simplify the handling of variable length fields since variable length data can be represented using pointers into a heap

IMPACT OF MEMORY RESIDENT DATA
Query Processing

- Sequential access is not significantly faster than random access in a memory resident database
- Hence, query processing techniques that take advantage of faster sequential access lose that advantage
  - Ex. Sort-merge join processing, which first creates sequential access by sorting the joined relations
  - The sorted relations could be represented easily in a main memory database using pointer lists
- Some relational operations can be performed very efficiently when relational tuples are implemented as a set of pointers to the data values
- The key idea is that because data is in memory, it is possible to construct appropriate, compact data structures that can speed up queries

IMPACT OF MEMORY RESIDENT DATA
Query Processing

- Query processors for memory resident data must focus on processing costs, whereas most conventional systems attempt to minimize disk access
- One difficulty is that processing costs can be difficult to measure in a complex data management system
- Costly operations (e.g., creating an index or copying data) must first be identified, and then strategies must be designed to reduce their occurrence
- Operation costs may vary substantially from system to system, so that an optimization technique that works well in one system may perform poorly in another

IMPACT OF MEMORY RESIDENT DATA Recovery

- Backups of memory resident databases must be maintained on disk or other stable storage to insure against loss of the volatile data
- Techniques such as commit processing require checkpointing, which brings the disk resident copy of the database more up-to-date
- This eliminates the need for the least recent log entries

IMPACT OF MEMORY RESIDENT DATA Recovery

- In a memory resident system, checkpointing and failure recovery are only reasons to access the disk-resident copy of the database
- Application transactions never require access to disk resident data
- Hence, disk access in a memory resident system can be tailored to suit the needs of the checkpointer alone
- Disk I/O should be performed using a very large block size, as large blocks are more effeciently written though they take longer
- Checkpointing should interfere with transaction processing as little as possible

IMPACT OF MEMORY RESIDENT DATA Recovery

- After a failure, a memory resident database manager must restore its data from disk resident backup and bring it upto-date
- Transferring data from the disks may take a long time
- One possible solution is to load blocks of the database ‘on demand’ until all of the data has been loaded
- Another possible solution to database restoration is to use disk striping or disk arrays
- The database is spread across multiple disks and read in parallel
- However in this case, there should be independent paths from disk to memory

IMPACT OF MEMORY RESIDENT DATA
Performance

- The performance of a main memory database manager depends primarily on processing time, and not on the disks
- Even recovery management, which involves the disks, affects performance primarily through the processor, since disk operations are normally performed outside the critical paths of the transactions
- This contrasts with models of disk-based ’ systems which count I/O operations to determine the performance of an algorithm
  - Ex. In conventional systems, making does not impact performance during normal system operation, so this component tends not to be studied carefully
  - In a MMDB, backups will be more frequent
- Thus the performance of backup or checkpointing algorithms is much more critical and studied more carefully

IMPACT OF MEMORY RESIDENT DATA
API and Protection

- In conventional DRDB's, applications exchange data with the database management system via private buffers
- In a MMDB, access to objects can be more efficient
- Applications may be given the actual memory position of the object, which is used instead of a more general object id
- A second optimization is to eliminate the private buffer and to give transactions direct access to the object
  - Ex. If a transaction is simple, most of its time may be spent copying bits from and to buffers
  - By cutting this out, the number of instructions a transaction must execute can be cut in half or more
  - Problem: Once transactions can access the database directly, they can read or modify unauthorized parts
  - Solution: Only run transactions that were compiled by a special database system compiler (checks for proper authorization)

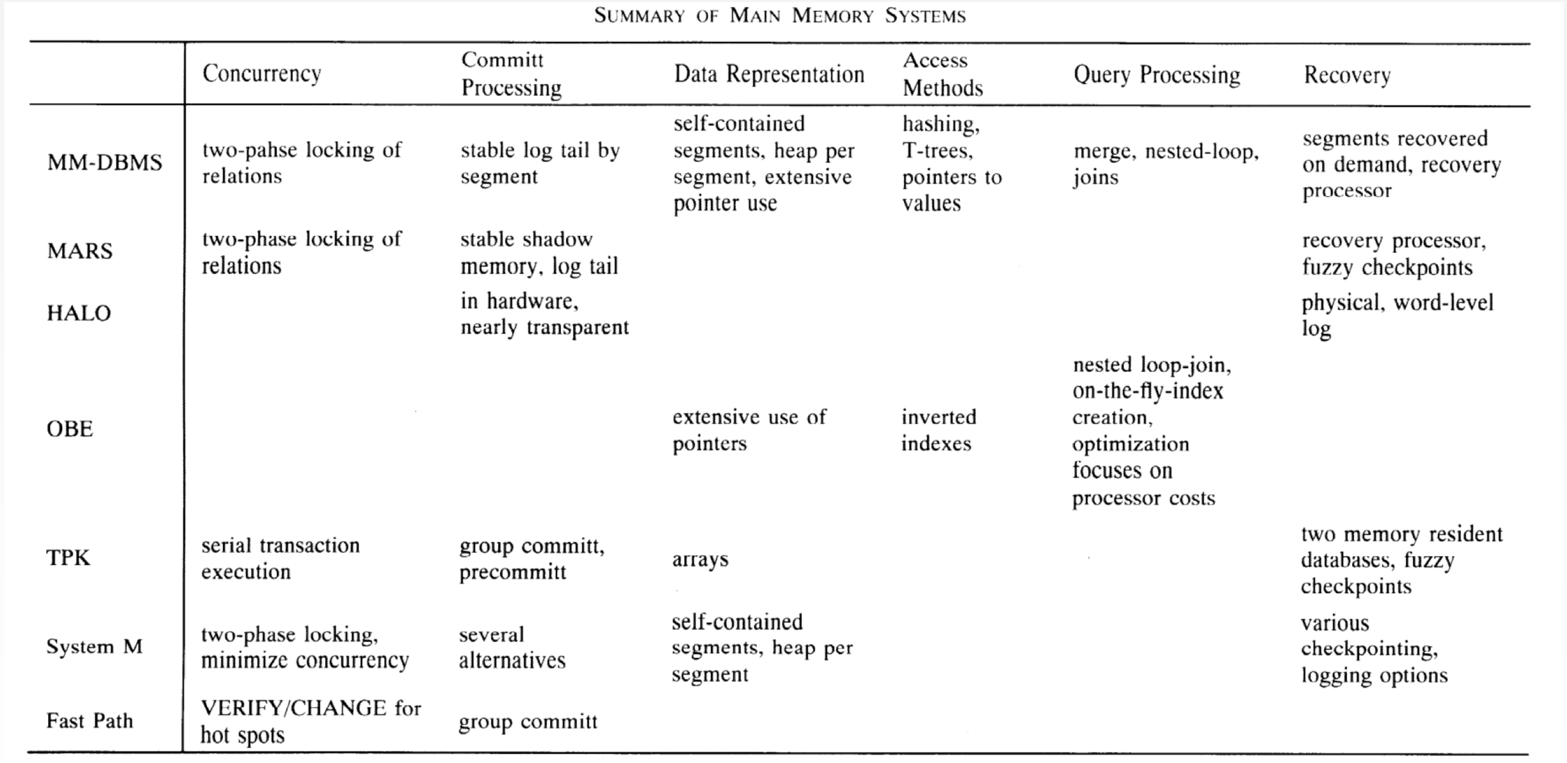
Data Clustering and Migration

- In a DRDB, data objects (e.g., tuples, fields) that are accessed together are frequently stored together, or clustered
- For instance, if queries often look at a "department" and all the "employees" that work in it, then the employee records can be stored in the same disk page as the department they work in
- In a MMDB there is no need to cluster objects
- If an object is to migrate to disk from memory, how and where should it be stored?
- Solutions
  - Users specify how objects are to be clustered if they migrate
  - The system determines the access patterns and clusters automatically

VIDEO

https://www.youtube.com/watch?v=j3knIXR-KHQ

SYSTEMS



Summary of Main Memory Systems

	Concurrency	Committ Processing	Data Representation	Access Methods	Query Processing	Recovery
MM-DBMS	two-pahse locking of relations	stable log tail by segment	self-contained segments, heap per segment, extensive pointer use	hashing, T-trees, pointers to values	merge, nested-loop, joins	segments recovered on demand, recovery processor
MARS	two-phase locking of relations	stable shadow memory, log tail				recovery processor, fuzzy checkpoints
HALO		in hardware, nearly transparent				physical, word-level log
OBE			extensive use of pointers	inverted indexes	nested loop-join, on-the-fly-index creation, optimization focuses on processor costs	
TPK	serial transaction execution	group committ, precommitt	arrays			two memory resident databases, fuzzy checkpoints
System M	two-phase locking, minimize concurrency	several alternatives	self-contained segments, heap per segment			various checkpointing, logging options
Fast Path	VERIFY/CHANGE for hot spots	group committ				

OBE

- Part of IBM's Office-By-Example (OBE) database project
- The system is designed to run on the IBM 370 architecture
- Its focus is on handling ad hoc queries rather than high update loads
- Data representation in the OBE system makes heavy use of pointers
- Query processing and optimization focus on reducing processing costs since queries do not involve disk operations
- Optimization techniques are also geared toward reducing or eliminating processor intensive activities

MM-DBMS

- The MM-DBMS system was designed at the University of Wisconsin
- Like OBE, MM-DBMS implements a relational data model and makes extensive use of pointers for data representation and access methods
- Variable length attribute values are represented by pointers into a heap, and temporary relations are implemented using pointers to tuples in the relations from which they were derived
- Index structures point directly to the indexed tuples, and do not store data values
- For recovery purposes, memory is divided into large self-contained blocks
- MM-DBMS uses two-phase locking for concurrency control

IMS/VS Fast Path

- IMS/VS Fast Path is a commercial database product from IBM which supports memory resident data
- Disk resident data are supported as well
- Each database is classified statically as either memory or disk resident
- Fast Path performs updates to memory resident data at commit time
- Transactions are group committed to support high throughput
- The servicing of lock requests is highly optimized to minimize the cost of concurrency control
- Record-granule locks are used
- Fast Path is designed to handle very frequently accessed data, since it is particularly beneficial to place such data in memory

MARS

- The MARS MMDB was designed at Southern Methodist University
- It uses a pair of processors to provide rapid transaction execution against memory resident data
- The MARS system includes a database processor and a recovery processor, each of which can access a volatile main memory containing the database
- A nonvolatile memory is also available to both processors
- The recovery processor has access to disks for the log and for a backup copy of the database
- The records are also copied into a nonvolatile log buffer
- Concurrency is controlled using two-phase locking with large lock granules (entire relations)

HALO

- Dedicated processors for recovery related activities such as logging and checkpointing?
- HArdware Logging (HALO) is a proposed special-purpose device for transaction logging
- It transparently off-loads logging activity from the processor(s) that executes transactions
- HALO intercepts communications between a processor and memory controllers to produce a word-level log of all memory updates
- Each time a write request is intercepted, HALO creates a log entry consisting of the location of the update and the new and old values at that address
- HALO includes a transaction identifier with each log record

TPK

- TPK is a prototype multiprocessor main-memory transaction processing system implemented at Princeton University
- TPK's emphasis is on rapid execution of debit/credit type transactions
- A simple data model consisting of records with unique identifiers
- Transactions may read and update records using the identifiers
- The TPK system of four types: input, consists of a set execution, output, of concurrent threads and checkpoint.
- Two copies of the database (primary and secondary) are retained in-memory
  - The primary copy supports all transaction reads and updates
  - The purpose of the secondary database is to eliminate data contention between the checkpoint and execution threads during the checkpoint operation

System M

- System M is a transaction processing testbed system developed at Princeton for main memory databases
- Like the TPK prototype, System M is designed for a transactional workload rather than ad hoc database queries
- It supports a simple record-oriented data model
- System M is implemented as a collection of cooperating servers (threads) on the Mach operating system
- Unlike TPK, System M is capable of processing transactions concurrently
- Since the focus of System M is empirical comparison of recovery techniques, a variety of checkpointing and logging techniques are implemented

QUESTIONS OR COMMENTS?

DISCUSSIONS

STRENGTHS

- A comprehensive explanation of in-memory databases along with a look into different in-memory database systems
- The paper provides insights into non-volatile memory and their impact

DISCUSSIONS

WEAKNESSES

- Insights into dynamic RAMs and virtual memory would have been useful
- An extensive performance evaluation was lacking; evaluation was mostly qualitative

THANK YOU!